Uniwersytet Jagielloński w Krakowie

Wydział Fizyki, Astronomii i Informatyki Stosowanej

Marcin Chamera

Nr albumu: 1153327

Metody detekcji i wyjaśniania anomalii w szeregach czasowych

Praca magisterska
na kierunku Informatyka stosowana

Praca wykonana pod kierunkiem
dr inż. Szymona Bobka
Zakład Technologii Gier

Kraków 2023

Abstrakt

W tej pracy przedstawione zostały metody detekcji i wyjaśniania anomalii w szeregach czasowych. Dokładnie przeanalizowano wyniki osiągane przed detektory wykorzystujące różnego rodzaju uczenie maszynowe, w celu znalezienia najlepszej metody detekcji, a także aby porównać ze sobą detektory uczenia głębokiego z tymi, które oparte są o uczenie zespołowe, z wykorzystaniem drzew decyzyjnych. Aby zrealizować ten plan, użyto zbiór danych zawierający transakcje z wykorzystaniem kart kredytowych, który został stworzony za pomocą generatora, a następnie na jego podstawie, w zależności od założeń danej metody ewaluacji, wytrenowano i zwalidowano w odpowiedni sposób przedstawione detektory. Ponadto, wnikliwej ewaluacji i dyskusji poddano dwie lokalne metody wyjaśniania, niezależne od badanego modelu, czyli SHAP i TS COIN. Końcowym zamierzeniem było ich porównanie i wybranie tej lepszej, pod względem czasu wykonywania, stabilności i zgodności obliczonych rankingów cech z tym oczekiwanym. TS COIN jest stworzonym na potrzeby tej pracy rozszerzeniem istniejącej metody COIN, które umożliwia jej użycie dla szeregów czasowych i detektorów sekwencyjnych, jednocześnie przyspieszając obliczenia dzięki wykorzystaniu programowania równoległego. Pokazano, że detektor gMLP, po uśrednieniu wyników dla wszystkich czterech wybranych metryk ewaluacyjnych, okazuje się być w tej kwestii najlepszy. Jednakże, analizując wyniki dla każdej z metryk z osobna, jest najlepszy tylko w wypadku średniej precyzji. Jest także jednym z wolniejszych detektorów pod względem czasu treningu i wnioskowania, dlatego jeśli istotnym byłoby, aby detektor był szybszy lub osiągał najlepszy wynik dla jednej z pozostałych trzech metryk, należałoby zastanowić się nad wyborem takich detektorów jak agregacja z drzewami decyzyjnymi, LSTM i XGBoost. Udowodniono także, że uczenie zespołowe z drzewami decyzyjnymi, dla wybranego zbioru danych, osiąga porównywalne wyniki do uczenia głębokiego. Na koniec pokazano, że TS COIN wypada lepiej od SHAP, zarówno w kwestii stabilności i zgodności obliczonych rankingów cech z tym oczekiwanym, jak i jeśli chodzi o czas wykonywania.

Abstract

This thesis presents methods of detection and explanation of anomalies in time series. The quality of detectors using different types of machine learning was thoroughly analysed to find the best detection method, and also to compare deep learning detectors with those based on ensemble learning using decision trees. To implement this plan, a dataset containing credit card transactions was used, which was created with the help of a generator, and then, depending on the pre-requirements of a given evaluation method, the detectors were trained and validated in an appropriate way. In addition, two local model-agnostic explainable methods, SHAP and TS COIN, were thoroughly evaluated and discussed. The end goal was to compare them and choose the better one in terms of execution time, stability and similarity of the calculated feature rankings to the expected one. TS COIN is an extension of the existing COIN method that was created for this work and allows the use of COIN for time series and sequential detectors while speeding up computations by using parallel programming. It has been shown that the gMLP detector, after averaging the results of all selected evaluation metrics, turned out to be the best. However, when analysing the results of each of the metrics separately, the detector is only the best with regard to average precision. It is also one of the slower detectors in terms of training and inference time, so if it was crucial for the detector to be faster, or to achieve the best result for one of the other three evaluation metrics, one should consider choosing detectors such as Bagging with decision trees, LSTM and XGBoost. It has also been proven that ensemble learning with decision trees, for the selected dataset, achieves comparable results to deep learning. Finally, TS COIN was shown to perform better than SHAP in terms of stability, compliance of the calculated feature rankings with the expected one, as well as in terms of execution time.

Chciałbym wyrazić szczerą wdzięczność mojemu promotorowi,
dr inż. Szymonowi Bobkowi, za jego nieocenione wskazówki
i zachętę w trakcie pisania tej pracy.

Spis treści

Rozdział 1
Wstęp

Uczenie maszynowe na stałe zagościło w naszych życiach. Pomaga nam osiągać rzeczy, których nie byliśmy wcześniej w stanie wykonać. Dzięki uczeniu maszynowemu jesteśmy w stanie dokonywać, między innymi, dokładnych prognoz, możemy generować nowe treści tekstowe i audiowizualne, czy też jesteśmy w stanie wykorzystywać systemy, którymi zarządzanie jest niebezpieczne dla człowieka, bądź niewykonalne. Algorytmy uczenia maszynowego podejmują za nas decyzje ważne, często zarządzając pokaźnymi sumami pieniędzy, starają się poprawiać nasz komfort życia, a nawet je chronić. Dlatego też, mimo bycia z reguły skutecznymi, ich decyzje powinny być zrozumiałe dla człowieka. Ponadto, będąc ludźmi, z natury jesteśmy po prostu ciekawi, co stało za nieoczekiwanym rezultatem danej akcji. Zwykle chcemy się też uczyć na błędach, aby nie powtarzać ich już w przyszłości. Między innymi, tym właśnie zajmuje się wyjaśnialna sztuczna inteligencja (ang. Explainable Artificial Intelligence, w skrócie XAI) – wyjaśnianiem wyników zwracanych przez modele uczenia maszynowego. Oprócz zagadnień związanych z etyką i bezpieczeństwem systemu, liczą się takie aspekty wyjaśnialności jak zrozumienie, w których momentach model podejmuje decyzje błędne i dlaczego, jak poprawić jego działanie i do jakich nowych wniosków można dojść. Model można uznać za lepiej wyjaśnialny, jeśli jego decyzje są relatywnie łatwiejsze do zrozumienia niż decyzje innego modelu.

Warto w tym miejscu jednak wspomnieć, że XAI nie jest nową dziedziną [72], a jedynie w ostatnich latach stała się bardziej popularna ze względu na coraz częstsze użycie bardziej skomplikowanych algorytmów. Już na początku badań związanych z SI, pojawiła się potrzeba wyjaśniania decyzji systemów [59]. Były to wówczas proste systemy SI, które zamiast uczenia maszynowego, wykorzystywały zasady (ang. rule-based), stworzone przez grupę ekspertów w danej dziedzinie. Dlatego też, wyjaśnienie decyzji systemów sprowadzało się do zwrócenia przez system informacji o zastosowanych zasadach i wykorzystanej wiedzy podczas jego tworzenia.

Jeśli chodzi zaś o współczesną wyjaśnialną SI, ma ona wiele aplikacji, między innymi:

- Wyjaśnianie decyzji o przyznaniu kredytu [8, 49] – wyjaśnialna SI zapewnia, że systemy zarządzania ryzykiem kredytowym nie są uprzedzone w stosunku do danej grupy, a decyzje podejmowane przez system są sprawiedliwe.

- Wyjaśnianie decyzji stojącej za detekcją szkodliwego nowotworu bądź też innego schorzenia [27, 73] – w obrazowaniu medycznym, wyjaśnialna SI pomaga doktorom i radiologom w podejmowaniu lepszych decyzji w diagnozowaniu chorób ze zdjęć czy wideo.

- Wyjaśnianie decyzji o zaklasyfikowania transakcji jako oszustwo [22, 56] – model

uczenia maszynowego to jedna z warstw w systemie do wykrywania oszustw w transakcjach. Wyjaśnienie, dlaczego dana transakcja wzbudziła alarm systemu, pomoże ekspertom decydującym o końcowej klasyfikacji lepiej zrozumieć przyczynę alarmu, co przerodzi się na precyzyjniejszą decyzję.

- Wyjaśnianie decyzji stojącej za predykcją, że dana maszyna zepsuje się w niedalekiej przyszłości [12, 53] – utrzymanie predykcyjne (ang. predictive maintenance) pozwala na wczesną detekcję potencjalnych problemów ze sprzętem, dzięki czemu pozwala zniwelować kosztowne naprawy i przestoje w funkcjonowaniu. Wyjaśnienie detekcji umożliwia podjęcie bardziej precyzyjnej i niezawodnej decyzji, a w wypadku stwierdzenia fałszywego alarmu pozwala oszczędzić czas serwisanta, i koszty związane ze zleceniem naprawy.

Wymienione wyżej przykłady wyjaśniania decyzji o detekcji nowotworu, oszustwa i przyszłej awarii maszyny mają wspólną cechę. Można je skategoryzować jako problemy wyjaśniania anomalii, a te dwa ostatnie jako problemy wyjaśniania anomalii w szeregach czasowych. W tej pracy uwaga jednak poświęcona będzie nie tylko samemu procesowi wyjaśniania, ale również wspomnianej detekcji. Dlatego też, aby zacząć opisywać temat wykrywania anomalii w szeregach czasowych, najpierw należy wyjaśnić, czym jest anomalia. Anomalia to wzorzec w danych, który nie jest zgodny z oczekiwanym zachowaniem. Anomalia jest czymś, co odbiega od tego, co jest standardowe, normalne lub oczekiwane. Jako synonim zwykle używa się terminu wartość odstająca, bądź też element odstający (ang. outlier). W tej pracy termin anomalia i element odstający będą używane zamiennie. Obserwacje, które nie odbiegają od normy, nazywa się wartościami typowymi, bądź też elementami typowymi (ang. inlier). Tak więc wykrywanie anomalii będzie oznaczało to samo, co wykrywanie wartości odstających, czyli wykrywanie nietypowości. W najprostszej formie jest to zadanie polegające na wykrywaniu anomalii punktowych, a więc pojedynczych instancji danych, które znacząco odstają od reszty. Jednak podczas pracy z szeregami czasowymi takie podejście może być naiwne. Szereg czasowy to sekwencja punktów danych, które są indeksowane zgodnie z kolejnością wyznaczoną przez czas. Wykrywanie anomalii w pod-sekwencjach szeregów czasowych powinno opierać się na konkretnym kontekście, w jakim znajduje się pod-sekwencja. W miarę możliwości, patrząc w przeszłość i próbując dostrzec czynniki wpływające na przyszłe punkty danych. Znalezienie przyczyny anomalii w szeregach czasowych może być zadaniem wymagającym, zwłaszcza gdy mamy do czynienia z punktami danych składającymi się nie tylko z jednej zmiennej (szereg czasowy jednowymiarowy), ale z wielu (szereg czasowy wielowymiarowy). Sezonowość, trendy i cykle to typowe wzorce, których występowanie należy zbadać, aby lepiej zrozumieć dane. Wykrywanie anomalii jest ważne, ponieważ pomaga identyfikować rzadkie zdarzenia, które często są prawie niemożliwe do wykrycia przez człowieka, gdy ma się krótki czas na reakcję i ogromne ilości dostępnych danych. Zwłaszcza jeśli mówimy o szeregach czasowych, gdzie zależności mogą być naprawdę złożone.

Różne typy anomalii i właściwości szeregów czasowych doprowadziły do opracowania wielu różnych algorytmów wykrywania anomalii. Każdy z nich ma mocne i słabe strony, dlatego dobranie odpowiedniej metody do danego zadania nie jest rzeczą łatwą. Jednak jeśli chodzi o metody wykrywania anomalii w dużych zbiorach danych, podejście oparte na uczeniu maszynowym jest zazwyczaj najlepszym sposobem na sprostanie temu wyzwaniu. Jedną z metodologii jest uczenie modelu predykcyjnego na podstawie zestawu danych historycznych z etykietami, a następnie użycie go do przewidywania etykiety

dla nowych danych. Ten proces nazywa się uczeniem nadzorowanym. Dla przykładu – do wykrywania oszustw, model próbuje przewidzieć klasę wyjściową i przypisać etykietę 0 lub 1, wskazując, czy transakcja jest legalna, czy też jest oszustwem – jest to zadanie klasyfikacji binarnej. Ewaluacja modelu odbywa się z wykorzystaniem zestawu testowego, z którymi wytrenowany model nie zetknął się wcześniej. Innym podejściem jest wykorzystanie metod uczenia nienadzorowanego, które nie wymagają etykietowania zestawu danych, bądź też wykorzystanie uczenia częściowo nadzorowanego, które warto zastosować, gdy istnieje tylko niewielka próbka zestawu danych z przypisaną etykietą.

Wyjaśnialna sztuczna inteligencja, szczególnie w zastosowaniu dla szeregów czasowych, jeśli chodzi o wachlarz dostępnych technik, jest dziedziną mniej zbadaną niż sama detekcja. Mimo to, na przestrzeni lat wykształciły się różne sposoby klasyfikacji metod dla wyjaśnialnej sztucznej inteligencji, między innymi ze względu na obszar interpretacji, element składowy modelu, który się interpretuje czy też to, czy w ogóle interpretuje się elementy składowe modelu, czy też jego wyniki. Wspomniana interpretacja, to nic innego jak właśnie omawiana wyjaśnialność, ponieważ w literaturze często terminy wyjaśnialna sztuczna inteligencja i interpretowalna sztuczna inteligencja stosuje się zamiennie [50]. Za jeden z ważniejszych podziałów można uznać podział modeli na tzw. białe skrzynki (ang. white-box) i czarne skrzynki (ang. black-box). Modele zaklasyfikowane jako białe skrzynki są modelami transparentnymi i interpretowalnymi ze względu na sposób, w jaki zostały skonstruowane. Będą to modele z prostszą strukturą, takie jak drzewo decyzyjne czy regresja liniowa, gdzie dla drzewa zasady podejmowania decyzji można wyczytać wprost z warunków utworzonych w węzłach drzewa, a dla regresji z wag przypisanych do każdej cechy wejściowej. Metody wyjaśniania konkretnego modelu, bądź też klasy modeli (ang. model-specific), będą zatem polegały na wewnętrznych mechanizmach modelu, w celu wyciągnięcia pewnych wniosków. Po drugiej stronie stoją modele zaklasyfikowane jako czarne skrzynki, które są modelami złożonymi, z wieloma parametrami i skomplikowanym sposobem działania. Dobrym przykładem są sieci neuronowe, które często posiadają wiele warstw i miliony, a nawet miliardy zmiennych parametrów. Modele czarnoskrzynkowe wyjaśnia się po ich treningu (ang. post-hoc), gdzie często stosuje się metody wyjaśniania, które nie badają wewnętrznych mechanizmów. Nie znaczy to jednak, że nie istnieją techniki wyjaśniania przeznaczone tylko dla konkretnej klasy modeli czarnoskrzynkowych. Metoda, która działa tylko dla sieci neuronowych też zalicza się do technik „model-specific". Przeciwieństwem są metody niezależne od badanego modelu (ang. model-agnostic), które z reguły analizują relację między cechami wejściowymi, a wynikiem. Ostatnim wymienionych sposobem klasyfikacji technik dla wyjaśnialnej SI będzie obszar interpretacji. Lokalne metody wyjaśnialnej SI pomagają w interpretacji indywidualnych predykcji, podczas gdy globalne metody pomagają w interpretacji ogólnego zachowania modelu dla całego zbioru danych.

Głównym celem tej pracy jest opracowanie, a także ocena metod oraz technik dla detekcji i wykrywania anomalii w szeregach czasowych. Rolą następnych rozdziałów będzie udzielenie odpowiedzi na następujące pytania:

- Jakie są istniejące metody detekcji anomalii w szeregach czasowych przy użyciu uczenia maszynowego i głębokiego?
- Jakie wyniki dla wybranego zbioru danych, osiągają detektory uczenia głębokiego w porównaniu do detektorów opartych o uczenie zespołowe z wykorzystaniem drzew decyzyjnych?

- Czy istnieje uniwersalnie najlepsza metoda detekcji anomalii?
- Jakie są istniejące metody wyjaśniania anomalii w szeregach czasowych?
- Czy istnieje lepsza metoda niż wykorzystanie biblioteki SHAP, jeśli chodzi o lokalne metody wyjaśnialnej SI, która będzie kompatybilna z różnymi detektorami sekwencyjnymi?

Struktura tej pracy jest zorganizowana w następujący sposób. W rozdziale 2 opisane zostały popularne metody detekcji anomalii w szeregach czasowych, wykorzystujące uczenie maszynowe, a także techniki związane z nierównowagą klas w zbiorze danych i użytecznymi metodami walidacji detektorów. Następnie, przedstawione zostały kroki, które zostały wykonane w celu stworzenia potoku do wykrywania anomalii w szeregach czasowych, ewaluacji wyuczonych detektorów i śledzenia wyników. Objaśniono również pojęcia związane z uczeniem maszynowym i głębokim, jak i zastosowane metryki ewaluacyjne. W rozdziale 3 opisano kilka z popularniejszych metod wyjaśnialnej sztucznej inteligencji, uwzględniając podział na te lokalne i globalne, a szczególną uwagę poświęcono bibliotece SHAP. Autorska implementacja TS COIN, razem z opisem działania bazowej metody COIN i wprowadzonych modyfikacji, została przedstawiona w rozdziale 4. W pierwszej części rozdziału 5 udokumentowane zostały wyniki testów dla metod detekcji. Pod uwagę wzięto wyniki dla wybranych metryk ewaluacyjnych, czas treningu i wnioskowania oraz optymalne wartości hiperparametrów znalezione dla detektorów. W drugiej części rozdziału 5 zaprezentowano wyniki testów dla wybranych metod wyjaśniania, czyli TS COIN i SHAP. Uwzględniony zostały rezultat testu dla wyniku odstawania z metody TS COIN, a wykorzystując wartości anormalnych wyników atrybutów z TS COIN wraz z wartościami SHAP, zmierzono stabilność metod oraz przy pomocy metryki NDCG, poprawność obliczonych rankingów cech. Na koniec przedstawiono wyniki czasów wykonywania dla obydwu metod. W rozdziale 6 omówione zostały wyniki z poprzedniego rozdziału, starając się jednocześnie udzielić odpowiedzi na postawione pytania badawcze z rozdziału 1. Objaśniono również napotkane wyzwania związane z testowaniem biblioteki SHAP, a także zaproponowano następne kroki w ramach kontynuacji porównania metod detekcji i wyjaśniania anomalii w szeregach czasowych.

Rozdział 2
Metody detekcji anomalii w szeregach czasowych

2.1 Popularne metody

Zarówno uczenie maszynowe, jak i jego podzbiór, jakim jest uczenie głębokie, są szeroko stosowane w problemach wykrywania anomalii szeregów czasowych [39]. Klasyczne, nadzorowane metody uczenia maszynowego obejmują maszynę wektorów nośnych (ang. Support Vector Machine, w skrócie SVM), K-najbliższych sąsiadów (ang. K-Nearest Neighbors, w skrócie KNN), regresję logistyczną lub drzewo decyzyjne. Jednak metody zespołowe (ang. ensemble learning) takie jak las losowy, a zwłaszcza te wykorzystujące wzmacnianie (ang. boosting), w większości przypadków przewyższają prostsze metody uczenia nadzorowanego. Metody zespołowe można uznać za tzw. state-of-the-art w tego typu problemach, szczególnie jeśli włożono odpowiednią pracę w staranną inżynierię cech zorientowaną wokół szeregów czasowych. Gdy nie ma, bądź też nie da się stworzyć, etykiet dla zbioru danych, metody takie jak las izolacyjny (ang. Isolation Forest) lub model mieszanin gaussowskich (ang. Gaussian Mixture Model) umożliwiają potraktowanie problemu wykrycia anomalii jako problem uczenia nienadzorowanego.

Metody uczenia głębokiego w przypadku dużej ilości danych mogą przewyższyć wspomniane wcześniej metody, a jednocześnie w znacznej części wypadków nie wymagają zaawansowanej inżynierii danych, ponieważ są często w stanie same zidentyfikować najbardziej użyteczne cechy. Zaczynając od metod nadzorowanych, od dawna znane są metody wykorzystujące podejście sekwencyjne, takie jak sieci rekurencyjne (ang. Recurrent Neural Network, w skrócie RNN) i ich odmiany, wykorzystujące komórki, rozwiązujące problem zaniku pierwszych wejść, np. LSTM (ang. Long Short-Term Memory). Zarówno wielowarstwowy perceptron (ang. Multilayer Perceptron, w skrócie MLP), jak i nowsze architektury oparte na transformatorach (ang. Transformer) mogą być również wykorzystywane do rozwiązania problemu wykrycia anomalii w sposób nadzorowany. Co więcej, wiele metod, pierwotnie stworzonych do zadań związanych z obrazami, można przekształcić w zadania oparte na szeregach czasowych. Przykładami są sieci konwolucyjne (ang. Convolutional Neural Network, w skrócie CNN) i ich bardziej zaawansowane odmiany jak na przykład InceptionTime [19], inspirowane architekturą Inception. W końcu metody nienadzorowane, takie jak autokodery (ang. Autoencoder), mogą być używane samodzielnie lub mogą być łączone z sieciami neuronowymi wykorzystującymi uczenie nadzorowane, albo poprzez włączenie danych wyjściowych z modelu nienadzorowanego do końcowego przewidywanego wyniku, albo

poprzez włączenie ich jako dodatkowego wejścia dla nadzorowanego modelu.

Jednym z wyzwań, jakie może pojawić się podczas stosowania nadzorowanego podejścia do wykrywania anomalii w analizie szeregów czasowych, jest nierównowaga klas. Anomalie są na ogół rzadkimi zdarzeniami, które mogą spowodować, że wytrenowany na niezbalansowanym zbiorze danych model będzie stronniczy w stosunku do elementów typowych. Z tego powodu, w wielu przypadkach, użyteczne jest zwiększenie liczebności niedostatecznie reprezentowanych klas, czyli tzw. nadpróbkowanie (ang. upsampling/oversampling) [29], bądź też zmniejszenie liczebności klasy większościowej (ang. undersampling) [29]. Innym podejściem do problemu niezbalansowanego zbioru danych jest ważenie klas [29, 41] na korzyść klasy mniejszościowej, czyli przywiązywanie większej uwagi do poprawnej predykcji modelu właśnie dla klasy mniejszościowej. Bez użycia wspomnianych metod, istotne informacje o rzadkich i nietypowych przypadkach mogą zostać utracone podczas treningu.

Jeśli chodzi o metody walidacji modelu, powszechnym podejściem, zapewniającym większą odporność modelu na zmiany w danych treningowych i dryf danych, jest stosowanie k-krotnego sprawdzianu krzyżowego (ang. k-fold cross-validation), czyli inaczej kroswalidacji. Kroswalidacja polega na dzieleniu zbioru danych na różne części i uczeniu modelu za każdym razem na innym podzbiorze. Metoda ta wskazuje jak dobrze klasyfikator jest w stanie generalizować. Jednak w przypadku danych szeregów czasowych, użycie kroswalidacji z losowym podziałem danych na część treningową i walidacyjną jest złym pomysłem. Jeśli wzorzec pojawi się w środkowej części zbioru treningowego uporządkowanego zgodnie z czasem, model prawdopodobnie wychwyci wzorzec, mimo że nie mógł on występować w początkowej części zbioru treningowego. Najlepszym podejściem jest tutaj zastosowanie łańcuchowania w przód (ang. forward-chaining), gdzie dla każdego podzbioru (ang. fold) kroswalidacji, część walidacyjna jest pobierana z fragmentu zbioru danych, znajdującego się później na osi czasu niż część treningowa.

2.2 Metodologia

W tej pracy podstawowy potok do wykrywania anomalii w szeregach czasowych, a także zbiór danych, który został wykorzystany do celów pracy, zostały zaczerpnięte z podręcznika online „Reproducible Machine Learning for Credit Card Fraud detection – Practical handbook" [39]. W swoim podręczniku autorzy dokładnie wyjaśnili, jak rozwiązać problem wykrywania oszustw związanych z kartami kredytowymi, który można traktować jako problem wykrywania anomalii w szeregach czasowych. Językiem programowania, który został wykorzystany do napisania kodu w ramach tej pracy, jest Python [20].

2.2.1 Zbiór danych

Uzyskanie dostępu do zbioru danych, który zawierałby rzeczywiste transakcje z wykorzystaniem kart kredytowych, ze względu na obowiązek zachowania poufności danych osobowych klientów, jest niemożliwe. Dlatego też autorzy zaproponowali użycie generatora, który tworzy zbiór danych symulujący prawdziwe dane. Powstały zbiór danych jest na tyle złożony, że naśladuje typowe zależności i wzorce występujące w rzeczywistych transakcjach, a jednocześnie jest wystarczająco przejrzysty jeśli chodzi o metody generowania, aby dotrzeć do pierwotnej przyczyny dlaczego niektóre

transakcje zostały zaklasyfikowane przez detektor jako oszustwo. Autorzy proponują również proces inżynierii cech, dostosowany do potrzeb analizy szeregów czasowych, agregowania istotnych cech w czasie i tworzenia nowych cech. W rezultacie każdy blok danych treningowych i walidacyjnych składa się z około 60 000 transakcji z 15 cechami wejściowymi. Autorzy proponują również zastosowanie uczenia głębokiego do zadania wykrywania anomalii oraz wykorzystanie kilku modeli z frameworka PyTorch [54]. W tym celu przedstawiono metodę tworzenia sekwencyjnych zbiorów danych, gdzie każda pod-sekwencja składa się z 5 ostatnich transakcji dla danego klienta.

2.2.2 Metryki

Na potrzeby niniejszej pracy wykorzystano również zestaw metryk ewaluacyjnych, zaproponowanych przez autorów podręcznika. Zdaniem autorów, dla tego typu problemu, należy użyć właśnie kilku różnych metryk, aby w bardziej niezawodny sposób ocenić wyniki modelu.

Pierwszą metryką będzie AUC ROC (ang. Area Under the Receiver Operating Characteristic curve), czyli metryka, która mierzy pole (AUC) pod wykresem krzywej charakterystyki roboczej odbiornika (ROC). Krzywa ROC to sposób oceny jakości predykcji klasyfikatora, najczęściej binarnego, który pokazuje zależność między czułością (ang. recall) i swoistością (ang. specificity). Metryki te opisuje się następującymi równaniami:

$$\text{czułość} = \frac{PP}{PP + FN}, \tag{2.1}$$

$$\text{swoistość} = \frac{PN}{PN + FP}, \tag{2.2}$$

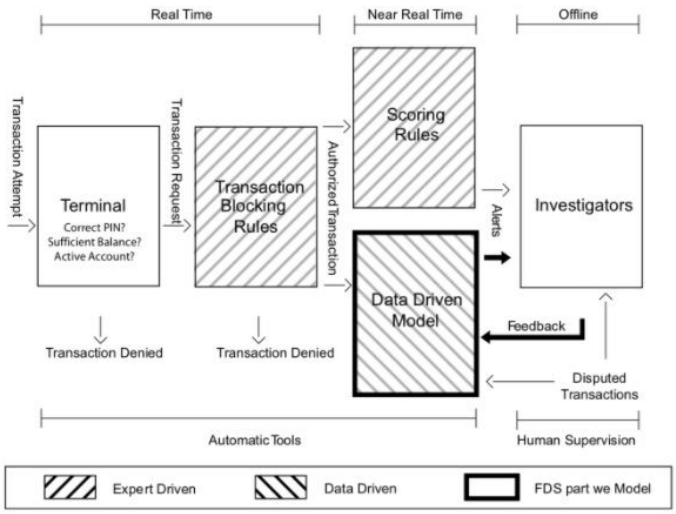
gdzie PP to przykłady prawdziwie pozytywne, FN to przykłady fałszywie negatywne, PN to przykłady prawdziwie negatywne, a FP to przykłady fałszywie pozytywne. Precyzując, dla krzywej ROC przedstawiamy zmienność czułości w zależności od 1 − swoistość dla różnych punktów odcięcia (ang. cut-off point). Jest to jedna z najczęściej używanych w literaturze metryk dla niezbalansowanych zbiorów danych, jednak czasem może dostarczyć mylące wyniki [51].

Dlatego też autorzy jako drugiej metryki, znowu niezależnej od wybranego punktu odcięcia, użyli średniej precyzji (ang. Average Precision, w skrócie AP). AP można przedstawić jako:

$$\text{AP} = \sum_n \left(R_n - R_{n-1}\right) P_n, \tag{2.3}$$

gdzie P_n i R_n to kolejno precyzja i czułość w n-tym punkcie odcięcia. Jest to metryka, która mierzy obszar pod krzywą precyzji w funkcji czułości (ang. Precision-Recall curve, w skrócie PR-curve), dlatego też nazwy średnia precyzja i AUC PR są sobie tożsame. Średnia precyzja jest według nowszych prac [58] lepszą metryką dla problemów klasyfikacji binarnej ze znacząco niezbalansowanym zbiorem danych.

Ostatnią z zaproponowanych w podręczniku metryk jest Card Precision top-k, w skrócie $CP@k$. Głównym założeniem dla tej metryki jest sprawdzenie jak dobrze detektor identyfikuje najbardziej połączone z oszustwami karty kredytowe. Pomysł na tę metrykę bazuje na tym, że predykcje detektora opartego o uczenie maszynowe nie są ostatecznym werdyktem w całym systemie wykrywania oszustw, a mają w praktyce służyć ekspertom z ostatniej warstwy systemu jako alarm, którego słuszność muszą ocenić. Schemat wspomnionego systemu ilustruje rysunek 1. Autorzy podręcznika



Rysunek 1: Diagram systemu do wykrywania oszustw, na bazie [39]

wychodzą z założenia, że liczba ekspertów weryfikujących alarmy z detektora jest ograniczona, tak samo ograniczony jest ich czas na ocenę. Z tego powodu, proponują wprowadzenie parametru k, który definiuje maksymalną liczbę kart możliwych do sprawdzenia w ciągu dnia. W oparciu o predykcje detektora, wybierana jest czołówka k kart najbardziej połączonych z oszustwami, a następnie dzięki prawdziwym etykietom, z czołówki wybierane są karty poprawnie zaklasyfikowane, które można oznaczyć jako TPC. Następnie, CP@k dla danego dnia obliczane jest poprzez podzielenie liczby kart TPC przez k. W przypadku, gdy zbiór testowy obejmuje większy okres niż jeden dzień, $CP@k$ będzie średnią wyników Card Precision top-k pośród wszystkich dni. Przy obliczaniu metryki CP@k dla danego dnia, karty poprawnie zaklasyfikowane jako powiązane z oszustwami są blokowane i usuwane z puli transakcji dla nadchodzących dni, co ma symulować blokadę takich kart w rzeczywistych systemach. W ramach tej pracy, zgodnie z decyzją autorów podręcznika, zdecydowano się wybrać liczbę 100 jako wartość parametru k.

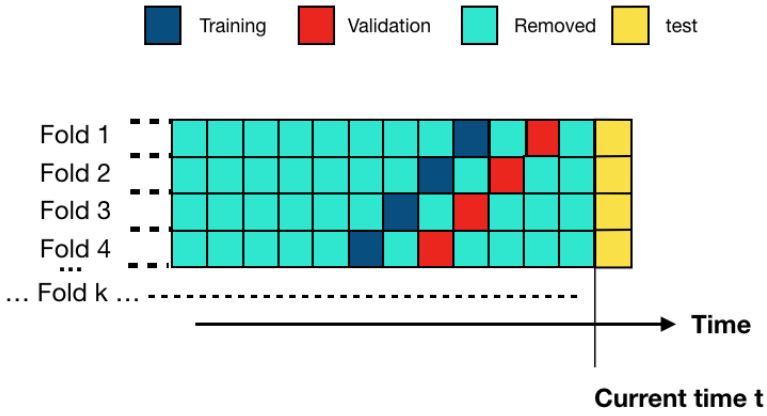
Ostatnią metryką, która została użyta w tej pracy w celu ewaluacji testowanych detektorów, ale jako pierwsza ze wspomnianych nie była wykorzystana przez autorów podręcznika, jest wynik F1 (ang. F1-score). Jest to średnia harmoniczna precyzji i czułości, którą można przedstawić jako:

$$F1 = 2 * \frac{precyzja * czułość}{precyzja + czułość`}. \tag{2.4}$$

Użycie średniej harmonicznej powoduje, że detektor uzyska dużą wartość wyniku F1 tylko, jeśli zarówno czułość i precyzja będą miały dużą wartość. Jest to jedyna metryka z wykorzystanych w tej pracy, która opiera się o wcześniej wybrany punkt odcięcia.

2.2.3 Walidacja

Jeśli chodzi o walidację modeli, autorzy wskazują, że najlepszym sposobem jest zastosowanie metody prequential validation. Jest ona rodzajem kroswalidacji dostosowanej pod szeregi czasowe. Autorzy proponują wydzielenie tzw. okresu opóźnienia między częścią treningową a walidacyjną, czyli wyraźnego odstępu w czasie pomiędzy ostatnią transakcją z części treningowej, a pierwszą transakcją z części walidacyjnej.



Rysunek 2: Walidacja przy użyciu metody prequential validation, na bazie [39]

Argumentują tę decyzję faktem, że prawdziwe etykiety dla przeprowadzonych transakcji nie są otrzymywane od razu, ponieważ w ostatnim etapie systemu wykrywania oszustw dla transakcji z udziałem kart kredytowych, grupa ekspertów musi podjąć ostateczną decyzję, czy transakcja jest oszustwem, czy też nie, a to wymaga czasu. W przypadku prequential validation, przy każdej zmianie części treningowej i walidacyjnej dla kolejnego k, części przesuwane są wstecz na osi czasu, o wybraną przez siebie odległość. W przeciwieństwie do niektórych proponowanych w literaturze podejść, gdzie części treningowe są coraz większe z każdą krotnością k dla sprawdzianu krzyżowego, autorzy podręcznika postanowili wybrać taką samą długość części treningowej dla kolejnych k, tak samo dla części walidacyjnej. Powodem jest dryf danych, którego potencjalne występowanie w danych autorzy uwzględnili w swojej metodzie, odrzucając w ten sposób starsze bloki danych transakcyjnych. Schemat metody prequential validation ilustruje rysunek 1.

2.2.4 Rozmiar grupy danych i liczba epok

W uczeniu głębokim rozmiar grupy danych (ang. batch size) to liczba przykładów, które przetwarzane są przez sieć neuronową, zanim wagi sieci zostaną zaktualizowane. Jest to jeden z ważniejszych hiperparametrów sieci neuronowych, który decyduje zarówno o czasie uczenia jak i skuteczności sieci. Im większy rozmiar grupy danych, tym sieć będzie się szybciej uczyć, ponieważ algorytm uczący będzie przetwarzał więcej przykładów na sekundę. Minusem większego rozmiaru grupy danych może być większa ilość pamięci potrzebna do pomyślnego załadowania wszystkich przykładów z grupy. Przyjęło się, że rozmiar grupy danych wybiera się spośród liczb będących potęgą dwójki. Takie też rozmiary grupy danych były testowane przez twórców podręcznika, to samo tyczy się detektorów wykorzystanych w tej pracy, które zostały zbudowane w oparciu o bibliotekę TSAI [52]. Pojawiają się jednak zdania ekspertów [5], że nie jest to obowiązkowa reguła, z racji, że obecne wersje najpopularniejszych frameworków dla uczenia głębokiego są w stanie również pracować z innymi rozmiarami grupy danych, równie efektywnie.

Epoka to proces przetworzenia przez sieć wszystkich przykładów ze zbioru treningowego. Jako hiperparametr, liczba epok będzie decydowała o tym, ile razy cały zbiór

treningowy zostanie przetworzony przez model, w ramach procesu uczenia. Jedną z technik pomocnych w znalezieniu optymalnej liczby epok jest wczesne zatrzymywanie (ang. early stopping). Wczesne zatrzymywanie będzie, co epokę, monitorowało wartość funkcji straty dla części walidacyjnej. Jeśli wartość nie zmaleje przez określoną liczbę epok, którą też można określić jako hiperparametr, trening modelu zostaje wstrzymany. Dopuszczalna liczba epok bez spadku funkcji straty, ustawiana dla wczesnego zatrzymywania, została nazwana przez autorów podręcznika cierpliwością (ang. patience). Wczesne zatrzymywanie znalazło użycie we wszystkich modelach z podręcznika opartych o uczenie głębokie. Mowa tu o uczeniu modeli z wykorzystaniem pojedynczego podziału zbioru danych na część treningową, walidacyjną i testową. W wypadku uczenia modeli z wykorzystaniem techniki prequential validation, wczesne zatrzymywanie nie może być użyte. Tak samo zostały przetestowane detektory z biblioteki TSAI. Dla detektorów wyuczonych z domyślnymi hiperparametrami, wykorzystano wczesne zatrzymywanie.

2.2.5 Podręcznik - przetestowane techniki

Następująca lista zawiera, które modele i techniki z tych zaproponowanych przez autorów podręcznika, zostały na potrzeby tej pracy odtworzone i wykorzystane na nowo:

- Drzewo decyzyjne
- Regresja logistyczna
- Losowy las
- Agregacja (ang. Bootstrap Aggregation, w skrócie Bagging) z drzewami decyzyjnymi
- XGBoost [10]
- Wielowarstwowy perceptron MLP
- LSTM [23]
- LSTM z komórką atencji (ang. attention) [2]
- CNN
- Autokoder
- Las izolacyjny
- Autokoder wraz z wielowarstwowym perceptronem, czyli podejście częściowo nadzorowane
- Zwiększenie liczebności klasy mniejszościowej w sposób losowy [29]
- Zmniejszenie liczebności klasy większościowej w sposób losowy [29]
- SMOTE (Synthetic Minority Oversampling Technique), czyli jedna z popularniejszych technik zwiększania liczebności klasy mniejszościowej, która generuje nowe dane z już istniejących, zamiast je tylko kopiować [3]
- Ważenie klas [29, 41]

2.2.6 Podręcznik - wyszukiwanie hiperparametrów

W podręczniku przedstawiono również proces wyszukiwania hiperparametrów, głównie przy użyciu metody przeszukiwania siatki (ang. grid search). Wyszukiwanie hiperparametrów, nazywane również strojeniem hiperparametrów, to proces, podczas którego porównuje się ze sobą różne kombinacje hiperparametrów dla określonego modelu, w celu znalezienia zestawu hiperparametrów, który zapewni najlepszy wynik dla wcześniej wybranych metryk ewaluacyjnych lub najniższej wartości funkcji straty. Metoda przeszukiwania siatki to koncepcyjnie najprostszy sposób poszukiwania optymalnych hiperparametrów, polegający na przetestowaniu wszystkich predefiniowanych kombinacji. Zapewnia to gwarancję znalezienia najbardziej optymalnego zestawu hiperparametrów z zadeklarowanej puli.

Z drugiej strony, metoda przeszukiwania siatki jest najbardziej czasochłonnym typem wyszukiwania, co sprawia, że nie jest najlepszym rozwiązaniem dla bardziej złożonych modeli, w których będzie więcej hiperparametrów. Dlatego też autorzy przedstawili drugą metodę wyszukiwania hiperparametrów, czyli przeszukiwanie losowe (ang. random search). Metoda losowego przeszukiwania sprawdza losowe kombinacje hiperparametrów w $x \in \mathbb{N}^*$ przebiegach, gdzie w każdym przebiegu losowo wybierana jest jedna, nieprzetestowana jeszcze kombinacja hiperparametrów z predefiniowanej puli. Jednak, aby nie duplikować niepotrzebnie pracy, podczas uczenia modeli z podręcznika dla celów tej pracy, hiperparametrów nie dostrajano, a jedynie wykorzystano optymalne kombinacje hiperparametrów znalezione przez autorów.

2.2.7 TSAI - wprowadzenie

Autorzy podręcznika pozostawili miejsce na dalszą eksplorację innych technik uczenia głębokiego, co stanowiło motywację do rozszerzenia listy testowanych detektorów. W tym celu wykorzystano bibliotekę TSAI [52], czyli jak reklamują ją jej twórcy ,,state-of-the-art bibliotekę uczenia głębokiego dla szeregów czasowych''. Opiera się na bibliotece FastAI [30], która z kolei została zbudowana na bazie frameworka PyTorch [54], co pomogło w ponownym wykorzystaniu szkieletu potoku przygotowywania danych z podręcznika. Z racji, że TSAI i FastAI posiadają też własne metody odpowiedzialne za kolejne etapy potoku przygotowywania danych, duża część metod stosowanych w potoku z podręcznika musiało zostać zastąpionych analogicznymi metodami z TSAI i FastAI.

2.2.8 TSAI - przetestowane detektory

Następujące 14 architektur, zaimplementowanych w TSAI, zostało wykorzystanych do pomyślnego wytrenowania modelu i uzyskania satysfakcjonujących wyników:

- LSTM [23]
- GRU [11]
- FCN [45]
- LSTM-FCN [36]
- MLSTM-FCN [37]

- GRU-FCN [16]
- ResNet [71]
- ResCNN [76]
- OmniScaleCNN [66]
- XCM [17]
- TST [75]
- TSiT, czyli adaptacja architektury ViT [15] dla szeregów czasowych, stworzona przez twórcę biblioteki TSAI
- gMLP [42]
- InceptionTime [19]

Podjęto również próbę wykorzystania innych dostępnych architektur i podejść zaproponowanych w bibliotece TSAI, takich jak mWDN [70] czy TSPerceiver [33], dla których trening zakończył się pomyślnie, jednak w trakcie ewaluacji osiągnęły niezadowalające wyniki, przez co nie będą uwzględnione w pozostałej części pracy. Najbardziej prawdopodobną przyczyną było niedostateczne zrozumienie interfejsu dostarczonego przez autorów TSAI dla wymienionych modeli oraz ich złe ustawienia początkowe.

2.2.9 TSAI - wyszukiwanie hiperparametrów

Trendem jest, że nowsze architektury modeli uczenia głębokiego są coraz bardziej złożone w kwestii działających w nich mechanizmów, co zwykle skutkuje dużym zakresem możliwych do ustawienia hiperparametrów. Biorąc również pod uwagę, że z reguły modele uczenia głębokiego wymagają dużej liczby parametrów do wyuczenia, trzeba poświęcić znaczącą ilość czasu na trening i walidację. Dlatego też, dla detektorów TSAI zdecydowano się na wykorzystanie algorytmu wyszukiwania losowego. Jak zaprezentowano we wpisie na blogu Alice Zheng [13], która opiera swoje obserwacje na pracy o wyszukiwaniu hiperparametrów przy użyciu wyszukiwania losowego [4], w wielu przypadkach wystarczy 60 przebiegów algorytmu wyszukiwania losowego, aby znaleźć prawie optymalny zestaw hiperparametrów. Taka liczba przebiegów została również wybrana do wyszukiwania hiperparametrów dla większości modeli z TSAI.

2.2.10 Optymalizatory i współczynnik uczenia

W uczeniu głębokim optymalizator to algorytm, który modyfikuje parametry sieci w celu zminimalizowania danej funkcji straty. Parametrami modelu, wykorzystującego uczenie głębokie, określa się jego wagi i obciążenia. Optymalizator pomaga poprawić wyniki modelu, modyfikując jego parametry i współczynnik uczenia (ang. learning rate).

Współczynnik uczenia jest hiperparametrem, który określa tempo, w jakim wybrany model będzie się uczył w kolejnych krokach uczenia. Jest to jeden z najbardziej wpływowych hiperparametrów, który może drastycznie zmienić wyniki modelu. Niski współczynnik uczenia powoduje powolne dążenie modelu do zbieżności, ponieważ wagi sieci są aktualizowane małymi wartościami. Z drugiej strony wysoki współczynnik

uczenia przyspieszy aktualizacje wag, co jednak może skutkować rozbieżnością modelu. Dlatego tak ważne jest ustawienie odpowiedniego tempa uczenia. Jednym ze sposobów jest testowanie różnych współczynników uczenia w celu znalezienia tego optymalnego, co też zrobili autorzy podręcznika.

Modele z podręcznika, które zostały wyuczone na potrzeby niniejszej pracy, wykorzystują algorytm optymalizacyjny Adam [38]. Adam jest rozszerzeniem optymalizatora stochastycznego spadku wzdłuż gradientu (ang. Stochastic Gradient Descent, w skrócie SGD) i jest jednym z najpopularniejszych dostępnych algorytmów optymalizacyjnych, między innymi dzięki dużej wydajności obliczeniowej i dobrym wynikom. Z oczywistych powodów dodanie wyszukiwania optymalnego współczynnika uczenia się do listy strojonych hiperparametrów wydłuży czas obliczeń, natomiast z reguły nie zagwarantuje znalezienia idealnej wartości. Na szczęście istnieją inne sposoby na znalezienie optymalnego współczynnika uczenia. Autor pracy [63] proponuje algorytm-planistę (ang. scheduler), który automatycznie będzie sterował współczynnikiem uczenia podczas treningu. Wspomniana tam polityka jednego cyklu polega na uczeniu modelu z wykorzystaniem współczynnika uczenia, który na początku wzrasta liniowo od małych wartości do zadeklarowanej wartości maksymalnej, a następnie liniowo maleje z powrotem do początkowej, niskiej wartości. Algorytm pomaga również wybrać wspomnianą wartość maksymalną. Dzięki temu algorytm polityki jednego cyklu pozwala na znacznie szybsze trenowanie złożonych modeli głębokiego uczenia. Z tego powodu jest to sugerowany i domyślny sposób znalezienia optymalnego współczynnika uczenia w bibliotece FastAI. Ze względu na to, że biblioteka TSAI czerpie wiele z FastAI, jest to również domyślny sposób doboru współczynnika uczenia podczas treningu z wykorzystaniem TSAI. Dlatego też modele oparte o bibliotekę TSAI, które zostały użyte na potrzeby tej pracy, zostały wytrenowane przy użyciu optymalizatora Adam, wspomaganego algorytmem polityki jednego cyklu.

2.2.11 Śledzenie wyników

Weights & Biases (w skrócie W&B) to platforma [6], która pomaga, zarówno amatorom uczenia maszynowego jak i profesjonalistom, w śledzeniu eksperymentów związanych z trenowaniem modeli, zapisywaniu metadanych (np. ustawionych hiperparametrów), wersjonowaniu zapisanych modeli i zbiorów danych, i wielu innych. Dla detektorów z podręcznika, platformę W&B użyto w celu:

- zapisania hiperparametrów ustawionych podczas trenowania modeli na nowo,
- zapisania wag wytrenowanych modeli,
- zapisania wyników z metryk ewaluacyjnych,
- zapisania czasu wykonywania fazy treningu i fazy testowania,
- zapisania wartości funkcji strat dla części treningowej i walidacyjnej, jak i wartości współczynnika uczenia, w każdej epoce − dla modeli uczenia głębokiego,

Powyższe punkty zostały również wykonane dla modeli opartych o bibliotekę TSAI. Dodatkowo dla tych modeli wykorzystano narzędzie oferowane przez platformę W&B o nazwie Sweeps, które umożliwia automatyzację procesu wyszukiwania optymalnych hiperparametrów, dostarczając przy tym zaawansowane wizualizacje. Funkcja ta umożliwiła monitorowanie na bieżąco procesu wyszukiwania hiperparametrów na stronie

internetowej platformy. W&B pozwala na dodanie znaczników (ang. tag) do uruchamianych procesów takich jak pojedynczy trening czy wyszukiwanie hiperparametrów. Dzięki temu wszystkie procesy uruchomione na platformie można w wybrany sposób filtrować i sortować, ponieważ zostały one wcześniej pogrupowane, właśnie za pomocą znaczników.

Rozdział 3
Wyjaśnialna sztuczna inteligencja

3.1 Popularne metody niezależne od modelu

3.1.1 Globalne

Wykres częściowej zależności (ang. Partial Dependence Plot, w skrócie PDP) to metoda, która pokazuje, jak dana cecha wpływa na średnią predykcję modelu uczenia maszynowego [26]. Głównym zamysłem jest to, aby zmienić konkretną cechę i zobaczyć, jak zmienia się predykcja, przy jednoczesnym zachowaniu wszystkich innych cech, z tymi samymi wartościami średnimi. W ten sposób można oddzielić wpływ jednej cechy od pozostałych. Na przykład, jeśli chcemy zobaczyć, jak data instalacji maszyny wpływa na predykcję detektora odnośnie nadchodzącej usterki, możemy stworzyć wykres średniej predykcji dla różnych dat instalacji, utrzymując wszystkie inne cechy (takie jak odczyty z sensorów czy data ostatniego przeglądu) w ich średnich wartościach. PDP może pokazywać nieliniowe i niemonotoniczne relacje miedzy cechą a predykcją.

Skumulowane efekty lokalne (ang. Accumulated Local Effects, w skrócie ALE) to kolejna z metod globalnych, niezależnych od modelu [1]. Idea jest podobna do PDP, ale zamiast utrzymywać wszystkie inne cechy, poza tą badaną, w ich średnich wartościach, uśrednia ich rozkład. W ten sposób, można rozważyć zależność cech i ich interakcje lepiej niż przy użyciu PDP, i uniknąć tworzenia sztucznych punktów danych, które nie są realistyczne. Tak samo jak PDP, ALE może pokazywać nieliniowe i niemonotoniczne relacje między cechą a predykcją.

Interakcja cech (statystyka H) to metoda, która określa ilościowo, w jakim stopniu predykcja jest wynikiem połączonych efektów cech, a nie każdej z nich pojedynczo [21]. Statystyka H przyjmuje wartości od 0 do 1, gdzie 0 oznacza brak interakcji, a 1 oznacza pełną interakcję. Można ją obliczyć dla dowolnej pary lub grupy cech i może być użyta do uszeregowania cech według siły ich interakcji. Statystyka H jednak nie pokazuje, w jaki sposób cechy wchodzą w interakcje ani jaki rodzaj interakcji mają (np. synergiczny lub antagonistyczny).

3.1.2 Lokalne

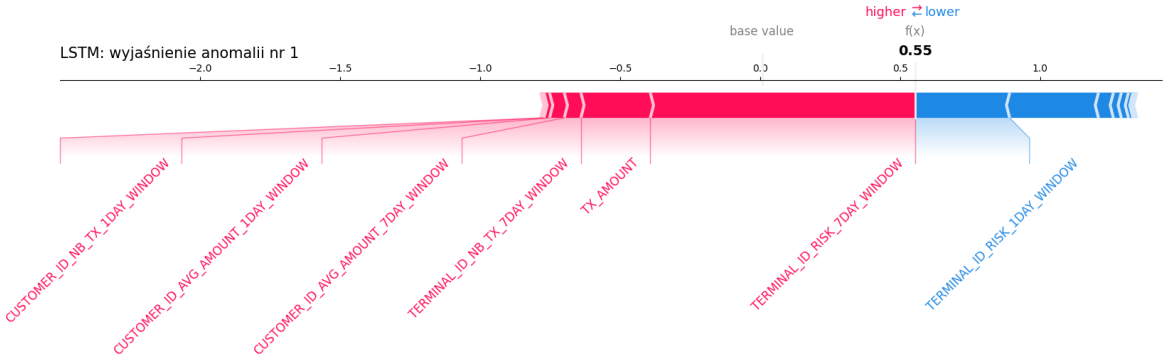
LIME (ang. Local interpretable model-agnostic explanations) to metoda, która pomaga w interpretacji pojedynczych predykcji dla złożonych modeli czarnoskrzynkowych [57]. LIME dopasowuje interpretowalny model w obszarze, w którym znajduje się pojedyncza predykcja, na przykład drzewo decyzyjne lub model regresji liniowej. Dzięki temu tworzona jest lokalna aproksymacja złożonego modelu czarnoskrzynkowego, nazywana modelem-surogatem. W metodzie LIME dąży się do tego, aby zminimalizować

różnicę pomiędzy wyjaśnieniami, które generuje, a faktycznymi predykcjami zwracanymi przez główny model, jednocześnie dbając o małą złożoność modelu-surogata.

Wartości Shapleya to pojęcie z teorii gier, które może posłużyć do wyjaśnienia pojedynczej predykcji modelu. W przypadku wyjaśnialnej SI koncept ten zakłada, że każda cecha wejściowa to gracz, a gracze biorą udział w grze, czyli w zadaniu predykcji dla pojedynczej obserwacji, przed jakim stoi model. Z kolei wypłata (ang. payout) na koniec gry, która trafia z powrotem do graczy, to różnica pomiędzy faktyczną predykcją dla danej obserwacji, a średnią predykcji dla wszystkich obserwacji. Obliczone wartości Shapleya będą natomiast informować, jak taką wypłatę rozdysponować między graczy, czyli cechy. Wartość Shapleya przypisana do danej cechy będzie mierzyć jej udział w predykcji, poprzez umieszczanie jej w różnych podzbiorach cech i porównywanie.

Lundberg i Lee wprowadzili nową metodę [47], która korzysta z wartości Shapleya. Została nazwana SHAP (ang. Shapley Additive exPlanations) i jest to przede wszystkim, ale nie tylko, metoda do wyjaśniania indywidualnych predykcji. Jest publicznie dostępna w formie rozbudowanej biblioteki i oprócz wspomnianej już umiejętności wyjaśniania pojedynczych predykcji, zawiera szereg globalnych metod wyjaśniania, które wykorzystują do tego agregację wartości Shapleya. SHAP łączy koncepty z wartości Shapleya i LIME, ponieważ wyjaśnienie dla wartości Shapleya jest dostarczane jako model liniowy, czyli metoda addytywnej atrybucji cech. Biblioteka SHAP dostarcza wiele algorytmów wyjaśniających (ang. explainer), które mają za zadanie oszacować wartości Shapleya. KernelExplainer jest algorytmem, który wyjaśnia modele, które zostały wytrenowane na danych tabelarycznych, czyli danych w formie dwuwymiarowej tabeli. W tym celu wykorzystuje metodę jądra (ang. kernel method). Metody jądra to grupa algorytmów, które używają liniowych klasyfikatorów w celu rozwiązywania nieliniowych problemów. Jądrem będzie funkcja podobieństwa dla wszystkich par obserwacji, obliczona przy użyciu iloczynów skalarnych. W pracy [46] został zaproponowany TreeSHAP, czyli algorytm wyjaśniający przeznaczony dla modeli opartych o drzewa takich jak drzewa decyzyjne, lasy losowe czy modele wykorzystujące wzmacnianie. GradientExplainer to kolejny algorytm wyjaśniający z biblioteki SHAP, który tym razem domyślnie przeznaczony jest dla danych obrazowych. Algorytm ten wykorzystuje dodatkowo metody SmoothGrad [62] i zintegrowanych gradientów (ang. Integrated Gradients) [64] w celu obliczenia wartości Shapleya dla każdego piksela. Ostatnim algorytmem wyjaśniającym z biblioteki SHAP, który będzie wymieniony w tej pracy, jest DeepExplainer. DeepExplainer jest implementacją Deep SHAP, algorytmu do obliczania wartości SHAP dla modeli uczenia głębokiego, który łączy algorytmy SHAP i DeepLIFT [61]. DeepExplainer, dla każdej warstwy modelu, oblicza gradient danych wyjściowych w stosunku do danych wejściowych. Te gradienty następnie są wykorzystywane w celu obliczenia udziału każdej cechy wejściowej w danych wyjściowych. Przy użyciu ważonej sumy gradientów, gdzie wagi są określone przez wartości SHAP, obliczany jest wkład każdej cechy. Do obliczenia wartości SHAP wykorzystywany jest algorytm wyjaśniający KernelExplainer w zmodyfikowanej wersji, która ma umożliwić wsparcie dla modeli uczenia głębokiego.

SHAP, oprócz implementacji algorytmów wyjaśniających, dostarcza również różnego rodzaju wykresy, które pomagają w wizualizacji wyjaśnień zarówno na poziomie pojedynczej predykcji jak i całego wyjaśnianego detektora. Jednym z najczęściej wykorzystywanych wykresów jest wykres sił (ang. force plot), który wizualizuje wpływ cech wejściowych na wynik jako siły. Wartość bazowa (ang. base value) to średnia wartość predykcji, obliczona z próbki zbioru treningowego o rozmiarze zdefiniowanym przez parametr nazywany rozmiarem danych tła (ang. background data size). Następnie,



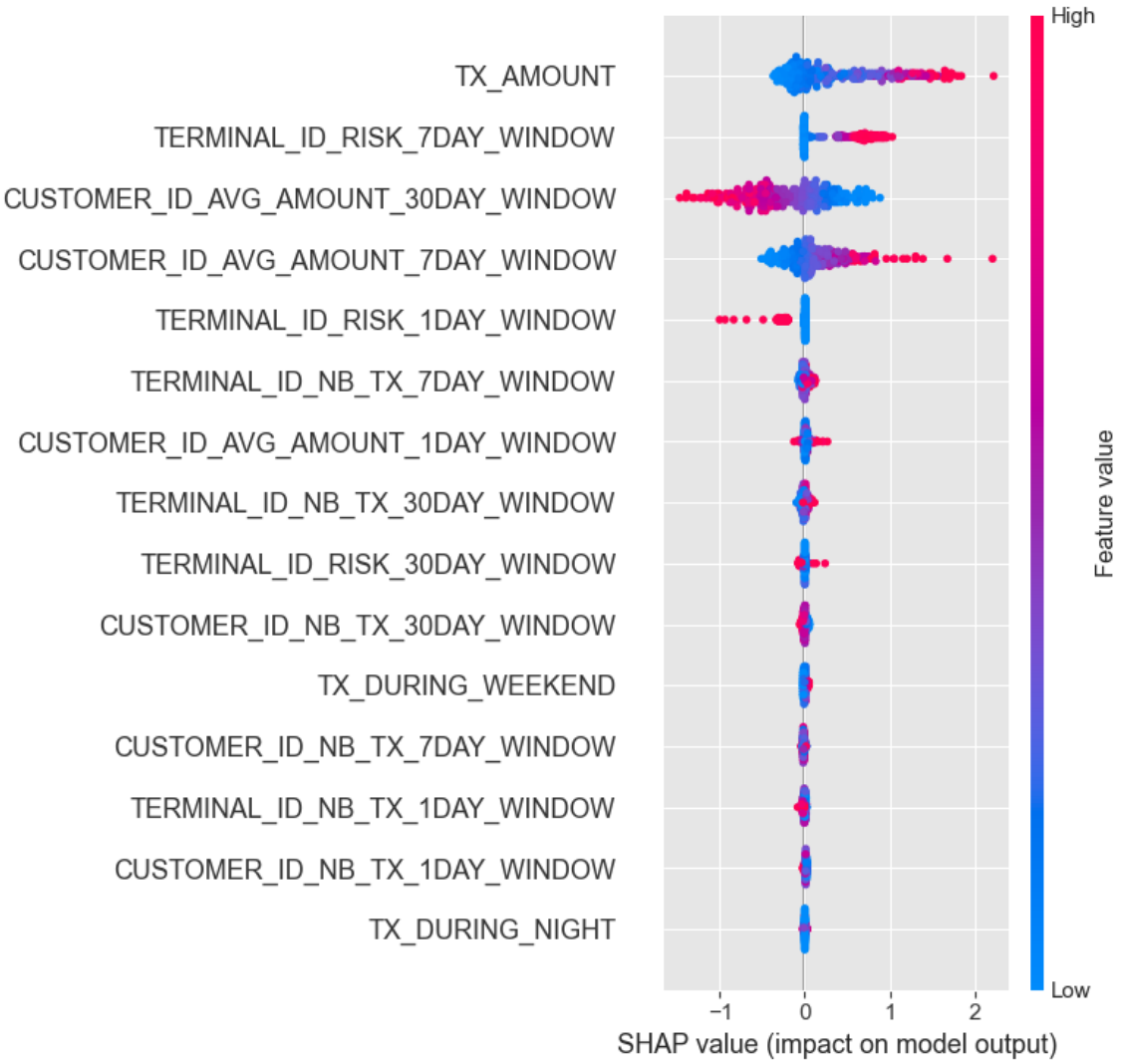
Rysunek 3: Przykładowy wykres sił stworzony przy pomocy biblioteki SHAP

każda wartość Shapleya, która odpowiada za konkretną cechę wejściową, popycha wartość bazową w prawą stronę, ku wyższemu wynikowi predykcji, o ile dana wartość Shapleya jest pozytywna. Jeśli jest ujemna, obniża wynik predykcji. Na ilustracji 3 przedstawiony został przykładowy wykres sił.

Przedstawia on wyjaśnienie dla pierwszej anomalii wykrytej przez detektor LSTM, który został wytrenowany na bazie podręcznika. Wartość bazowa, wynosząca 0.00662505, została zaokrąglona na wykresie do zera, a prawdopodobieństwo anomalii według użytego detektora wynosi 0.55. Największy wpływ na wysoki wynik predykcji ma cecha TERMINAL_ID_RISK_7DAY_WINDOW, z kolei największy wpływ na niski wynik predykcji ma cecha TERMINAL_ID_RISK_1DAY_WINDOW.

Kolejnym wykresem, który można utworzyć przy pomocy biblioteki SHAP, jest wykres podsumowujący (ang. summary plot), który pozwala na wizualizację wpływu cech (ang. feature importance) wraz z efektami cech (ang. feature effects). Cechy wejściowe, przedstawione na osi Y, posortowane są w kolejności ich ogólnego wpływu na predykcje detektora, malejąco. Natomiast każdy punkt na osi X przedstawia obliczoną wartość Shapleya dla danej cechy na osi Y, gdzie kolor reprezentuje jej wartość − wysokie wartości są bardziej czerwone, a mniejsze wartości bardziej niebieskie. Nagromadzone punkty, tzn. podobne wartości Shapleya dla danej cechy będą rozszerzały się wzdłuż osi Y, dzięki czemu można oszacować dystrybucję wartości dla danej cechy. Na ilustracji 4 przedstawiony został przykładowy wykres podsumowujący, stworzony na podstawie wyjaśnień predykcji detektora LSTM z podręcznika.

Wspomniane rodzaje wykresów to nie jedyne, które udostępnia biblioteka SHAP. Można znaleźć w niej jeszcze m.in. wykres przedstawiający sam wpływ cech (SHAP Feature Importance), czy też wykres zależności (SHAP Dependence Plot).



Rysunek 4: Przykładowy wykres podsumowujący, stworzony przy pomocy biblioteki SHAP

Rozdział 4
TS COIN

4.1 COIN

COIN (Contextual Outlier Interpretation) [44] to niezależna od modelu metoda, która służy do wyjaśniania anomalii wykrytych przez detektor. Wykorzystuje tzw. kontekst do wyjaśnienia i weryfikacji wykrytej anomalii. Jako wynik zwraca: wynik odstawania (ang. outlierness score), czyli pojedynczą liczbę dla każdej wykrytej anomalii – im liczba większa, tym silniej wskazuje na poprawność zaklasyfikowania danej obserwacji jako anomalii; anormalny wynik atrybutu (ang. abnormal attribute score), który przypisuje wagę każdej cesze wejściowej dla danej obserwacji – waga ta wskazuje, w jakim stopniu dana cecha przyczyniła się do określenia tej obserwacji jako element odstający.

Kolejne kroki wykonywane w COIN można scharakteryzować w następujący sposób: dla każdego elementu odstającego rozpoznanego przez detektor odnajdywany jest kontekst, czyli najbliżsi sąsiedzi z grupy elementów typowych. Następnie kontekst jest dzielony na klastry. W celu określenia optymalnej liczby klastrów L, autorzy proponują użycie miary prediction strength [68], gdzie głównym zamysłem jest potraktowanie problemu klasteryzacji jako nadzorowanego problemu klasyfikacji, w którym oszacowywane są rzeczywiste etykiety klas. Jeśli chodzi o sam proces klasteryzacji, twórcy w swojej pracy nie sugerują jednego, konkretnego algorytmu. Jednakże, w implementacji metody stworzonej przez jednego z autorów pracy [43], użyty został algorytm K-Means. Klastry, których rozmiar nie przekracza z góry ustalonego punktu progowego, są odrzucane. Autorzy w pracy proponują ustalenie punktu progowego o wartości 0.03 całkowitego rozmiaru kontekstu:

$$|\mathcal{C}_{i,l}| \leqslant 0.03 \cdot |\mathcal{C}_i|\,, \tag{4.1}$$

gdzie $|\mathcal{C}_i|$ to liczba elementów kontekstu dla anomalii i, a $|\mathcal{C}_{i,l}|$ to liczba elementów utworzonego klastra o indeksie $l \in \{1, \ldots, L\}$. Natomiast w samej implementacji to podejście uproszczono, wybierając dla punktu progowego domyślną wartość jako liczbę naturalną. W kolejnym kroku, COIN traktuje każdy klaster z osobna, nadpróbkując element odstający do takiej liczby podobnych obserwacji, aby liczba elementów typowych w klastrze była równa liczbie elementów odstających po nadpróbkowaniu. Wspomniane podejście rozwiązuje problem braku balansu pomiędzy liczbą elementów typowych i odstających. Autorzy sugerują w swojej pracy użycie metody syntetycznego próbkowania [28], które znalazło się również we wspomnianej implementacji. Następnie, dla każdego nieodrzuconego klastra z nadpróbkowanym elementem odstającym, trenowany jest lokalny klasyfikator binarny. W pracy autorzy sugerują użycie maszyny wektorów nośnych. Argumentują ten wybór dobrą zdolnością maszyny wektorów nośnych do izolacji elementów odstających. Aby jednak zachować główną ideę takiego lokalnego

klasyfikatora, czyli interpretowalność, wybrali jej liniowy wariant. Liniowe maszyny wektorów nośnych są interpretowalne z racji, że każda cecha wejściowa ma przypisaną wagę, która bezpośrednio wpływa na predykcję modelu. W ostatnim kroku metody COIN, lokalne klasyfikatory oraz informacje o elementach klastrów są wykorzystywane do obliczenia wyniku odstawania i anormalnego wyniku atrybutu. Anormalny wynik atrybutu a_m dla anomalii o_i przedstawia się jako:

$$s_i\left(a_m\right) = \left(1/\left|\mathcal{C}_i\right|\right) \sum_l \left|\mathcal{C}_{i,l}\right| s_{i,l}\left(a_m\right), \tag{4.2}$$

czyli sumę ważoną wyników dla a_m pośród wszystkich klastrów. Wynik w odniesieniu do klastra $\mathcal{C}_{i,l}$ definiowany jest jako:

$$s_{i,l}\left(a_m\right) = \left|w_{i,l}[m]\right| / \gamma_{i,l}^m, \tag{4.3}$$

gdzie $w_{i,l}[m]$ jest m-tą wagą z wektora wag lokalnego klasyfikatora $g_{i,l}$, a $\gamma_{i,l}^m$ jest średnim dystansem wzdłuż m-tej osi pomiędzy instancją w klastrze $\mathcal{C}_{i,l}$ i jej najbliższymi sąsiadami. Wynik odstawania dla anomalii o_i przedstawia się jako:

$$d\left(\mathbf{o}_i\right) = \left(1/\left|\mathcal{C}_i\right|\right) \sum_l \left|\mathcal{C}_{i,l}\right| d_l\left(\mathbf{o}_i\right) / \gamma_{i,l}, , \tag{4.4}$$

czyli sumę ważoną wyników dla a_m pośród wszystkich klastrów, znormalizowaną przez $\gamma_{i,l}$, które jest średnim dystansem od instancji do jej najbliższego sąsiada w $\mathcal{C}_{i,l}$. Wynik w odniesieniu do klastra $\mathcal{C}_{i,l}$ definiowany jest jako:

$$d_l\left(\mathbf{o}_i\right) = \left|g_{i,l}\left(\mathbf{o}_i\right)\right| / \left\|\mathbf{w}_{i,l}\right\|_2, \tag{4.5}$$

gdzie $g_{i,l}\left(\mathbf{o}_i\right)$ jest wynikiem funkcji decyzyjnej klasyfikatora dla anomalii, czyli odległością anomalii od granicy decyzyjnej klasyfikatora, która to rozdziela klasy.

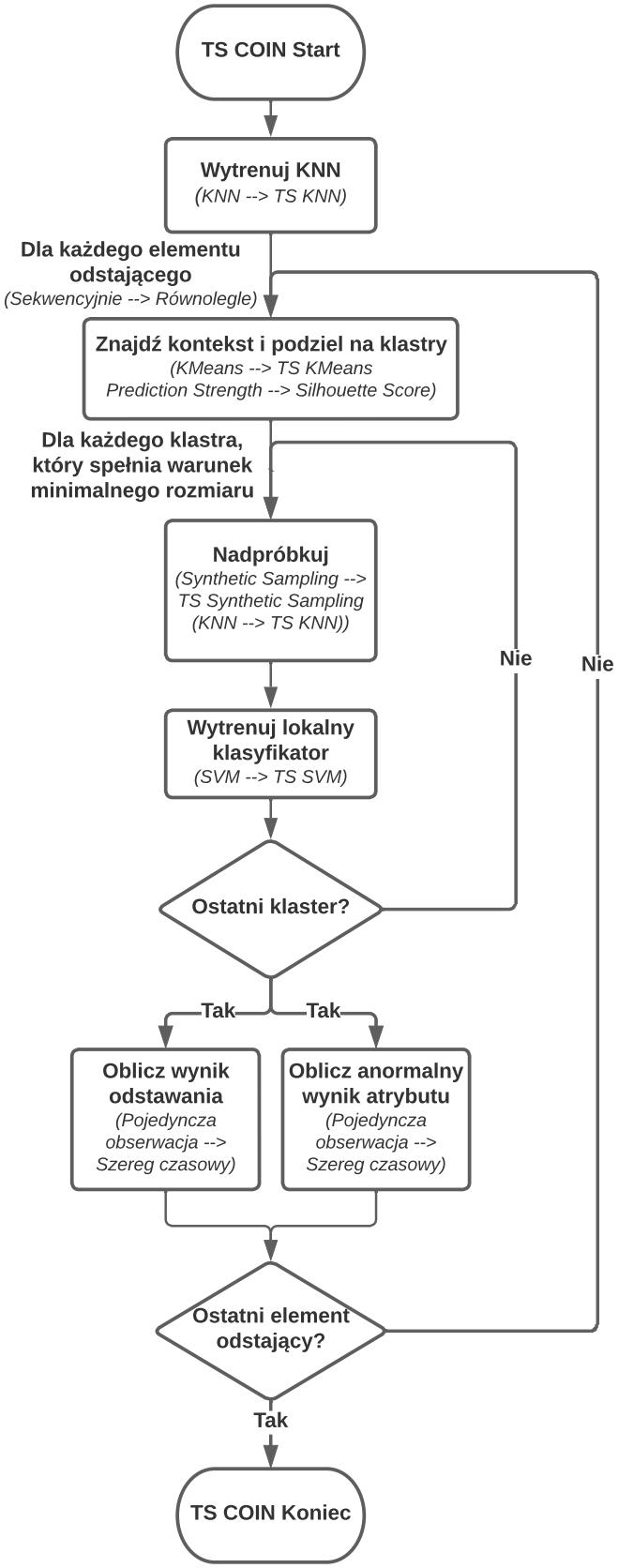
4.2 Uwzględnienie szeregów czasowych

COIN został przygotowany przez jego autorów do analizy indywidualnych obserwacji, które nie są od siebie zależne w czasie. Jednak w przypadku szeregów czasowych anomalią może być nie pojedynczy punkt, ale pewna pod-sekwencja – fragment szeregu czasowego. Ponadto, oryginalna implementacja COIN nie może być zastosowana do detektorów sekwencyjnych, które wymagają trójwymiarowych danych wejściowych. Ze względu na to ograniczenie, w tej pracy proponowana jest implementacja metody COIN, która będzie dostosowana do pracy z szeregami czasowymi i modelami sekwencyjnymi – TS COIN, gdzie TS jest oznaczeniem szeregu czasowego (ang. time series). Celem tej implementacji było zmienienie tylko niezbędnych części metody COIN, aby obsłużyć dane szeregów czasowych. Schemat działania TS COIN, wraz z informacjami o zmianach w poszczególnych krokach metody w porównaniu do oryginalnej implementacji, ilustruje rysunek 5.

W oryginalnej implementacji COIN wybrano bibliotekę scikit-learn [55], w celu użycia gotowych modułów z algorytmami uczenia maszynowego. Zarówno dla początkowego algorytmu KNN, który uczony jest na wszystkich elementach typowych w celu późniejszego znalezienia kontekstu dla elementu odstającego, jak i dla algorytmu KMeans, który dzieli kontekst na klastry, a także dla SVM, który pełni funkcję lokalnego

klasyfikatora, wykorzystano implementację z scikit-learn. W TS COIN, wspomniane implementacje zostały zastąpione ich odpowiednikami z biblioteki TSLearn [67]. Jest to biblioteka, która dostarcza narzędzia związane z uczeniem maszynowym, dla analizy szeregów czasowych. Wskutek tego, wykorzystano implementację KNN dla szeregów czasowych, dostępną jako klasa KNeighborsTimeSeries. Podobne zmiany zaszły wśród pozostałych algorytmów. Dla modelu KMeans wykorzystano implementację dla szeregów czasowych, dostępną jako klasa TimeSeriesKMeans, a dla klasyfikatora SVM implementację dostępną jako klasa TimeSeriesSVC. Dodatkowo, zrezygnowano z algorytmu prediction strength odpowiedzialnego za znalezienie optymalnej liczby klastrów i zastąpiono go algorytmem silhouette score, którego implementacja jest dostępna w TSLearn, pod postacią funkcji silhouette_score.

W oryginalnej implementacji COIN, obliczenia dla każdego elementu odstającego wykonywane są sekwencyjnie. To znaczy, że cały proces składający się ze znajdywania kontekstu, podziału na klastry, nadpróbkowania, uczenia klasyfikatora, a następnie obliczania wyniku odstawania i anormalnego wyniku atrybutu, jest powtarzany po sobie dla każdego z elementów odstających. W TS COIN wykorzystano fakt, że większość obliczeń w tej metodzie jest wykonywanych niezależnie pomiędzy elementami odstającymi. Dlatego też, tuż na początku metody TS COIN, trenowany jest raz model KNN, a następnie wszystkie obliczenia wykonywane są w sposób równoległy. Biorąc pod uwagę liczbę dostępnych procesorów logicznych, jakie oferował użyty komputer, zdecydowano się na wykonywanie obliczeń dla 10 elementów odstających naraz, gdzie jeden procesor logiczny wykonuje obliczenia dla jednego elementu odstającego. Wliczając jeszcze dodatkowy procesor logiczny, który uruchamia blok obliczeń równoległych, wykorzystany komputer pozostaje z ostatnim dostępnym procesorem logicznym, co w efekcie nie unieruchamia systemu na czas obliczeń i pozwala na wykonywanie innych akcji związanych z obsługą maszyny. W celu zrównoleglenia obliczeń użyto bibliotekę Joblib [35], która pozwala uruchamiać funkcje języka Python jako równoległe zadania (ang. parallel jobs). Jeśli chodzi zaś o to, w jakiej formie metoda TS COIN oczekuje danych wejściowych, muszą być one wprowadzone jako tablice Numpy [25]. Numpy, jako najpopularniejsza biblioteka do obliczeń numerycznych w języku Python, dostarcza również odpowiednich instrukcji, których użycie pozwoliło na modyfikację pozostałych fragmentów metody TS COIN, w celu dopasowania jej do przetwarzania szeregów czasowych. Finalna implementacja TS COIN jest dostępna w repozytorium na GitHub [9].



Rysunek 5: Schemat działania TS COIN, zmiany w porównaniu do oryginalnej implementacji są wymienione w nawiasach

Rozdział 5
Ewaluacja

W tej części pracy przedstawione zostaną wyniki ewaluacji testowanych detektorów jak i wyniki eksperymentów przeprowadzonych przy użyciu metod TS COIN oraz SHAP.

Eksperymenty przeprowadzono na komputerze wyposażonym w procesor AMD Ryzen 5 5600H (6x 3.3 GHz), 32 GB pamięci RAM DDR4 oraz kartę graficzną NVIDIA GeForce RTX 3060 (6 GB GDDR6).

5.1 Detekcja anomalii

W tej sekcji pracy porównane zostaną detektory zarówno z podręcznika jak i zbudowane przy pomocy biblioteki TSAI.

5.1.1 Metryki ewaluacyjne

Pierwszym testem, który ma za zadanie sprawdzić jakość detektorów i zastosowanych technik, będzie porównanie ich wyników dla metryk ewaluacyjnych. Dla każdej z metryk przedstawionych zostanie 10 detektorów i technik z najlepszym wynikiem. Podczas wyszukiwania hiperparametrów dla modeli TSAI, użyto metody losowego przeszukiwania z 60 przebiegami, w celu zminimalizowania funkcji straty dla części walidacyjnej, a w każdym przebiegu użyto prequential validation jako metody walidacji. Wyjątkiem w kwestii obszaru wyszukiwania hiperparametrów jak i liczby przebiegów są detektory OmniScaleCNN oraz ResNet. Dla detektora ResNet, w implementacji dostępnej w bibliotece TSAI, nie ma możliwości ustawienia hiperparametrów ściśle powiązanych z tą architekturą, dlatego też wyszukiwanie hiperparametrów zawężono do rozmiaru grupy danych i liczby epok. Odnosząc się zaś do detektora OmniScaleCNN, autor architektury we własnej implementacji dostępnej na repozytorium GitHub [65] wyjaśnia, że OmniScaleCNN działa dobrze z domyślnym ustawieniem hiperparametrów i nie ma potrzeby wyszukiwania tych optymalnych. Podobnie jak w przypadku detektora ResNet, dla OmniScaleCNN zakres wyszukiwania zawężono do rozmiaru grupy danych i liczby epok. Liczbę przebiegów wyszukiwania hiperparametrów dla obu tych detektorów ustawiono na 24 ze względu na relatywnie wąski obszar przeszukiwania. W praktyce dla tych dwóch detektorów, właśnie ze względu na wąski obszar przeszukiwania, metoda losowego przeszukiwania zamieniła się w metodę przeszukiwania siatki. Optymalne hiperparametry dla modeli z podręcznika zostały wybrane na podstawie przedstawionych tam wartości, otrzymanych w wyniku jednego z zastosowanych wyszukiwań. Wyjątkiem będą detektory wykorzystujące uczenie nienadzorowane, czyli autokoder, las izolacyjny

i autokoder połączony z siecią MLP. Autorzy podręcznika nie zdecydowali się na wyszukiwanie hiperparametrów dla tych detektorów. Dlatego też, dla wspomnianych detektorów uczenia nienadzorowanego, aby nie dopuścić do przetrenowania na danych uczących (ang. overfitting), zastosowano technikę wczesnego zatrzymywania, która wyznaczyła optymalną liczbę epok. Finalnie, dla wszystkich przedstawionych w tej pracy detektorów, metryki ewaluacyjne zostały policzone na podstawie wyników dla zbioru testowego.

5.1.1.1 Wynik F1

Przypominając, wynik F1 jest rodzajem metryki opartym o punkt odcięcia. Zgodnie z powszechną praktyką, zdecydowano się wybrać wartość 0.5 jako odpowiedni punkt odcięcia. Oznacza to, że każda predykcja detektora, oparta o prawdopodobieństwo przynależenia do klasy transakcji-oszustw, która wyniesie więcej niż 0.5, zostanie zaklasyfikowana właśnie jako oszustwo. Warto jednak pamiętać, że jest to tylko wartość umowna, a dla faktycznego użycia w systemie wykrywania oszustw należałoby przetestować otrzymywane wyniki dla różnych punktów odcięcia w celu wybrania tego najlepszego. Szczegółowe wyniki znajdują się w tabeli 1.

Model	Źródło	Wynik F1	Technika dla niezbalansowanych zbiorów danych
LSTM	TSAI	0.731	Nie dotyczy
XGBoost	podręcznik	0.730	Brak
gMLP	TSAI	0.729	Nie dotyczy
Las losowy	podręcznik	0.714	Brak
Drzewo decyzyjne	podręcznik	0.713	Brak
TSiT	TSAI	0.708	Nie dotyczy
TST	TSAI	0.707	Nie dotyczy
GRU	TSAI	0.692	Nie dotyczy
ResCNN	TSAI	0.688	Nie dotyczy
ResNet	TSAI	0.682	Nie dotyczy

Tabela 1: Detektory i zastosowanie techniki dla niezbalansowanych zbiorów danych z najwyższym wynikiem F1

Wartości wyniku F1, dla trzech najlepszych detektorów, różnią się między sobą marginalnie, więc przy nieco innym rozkładzie danych, kolejność detektorów mogłaby ulec zmianie. Gdy spojrzy się jednak na wynik następnego w kolejności detektora, jest już on znacząco mniejszy. Oznacza to, że można by uznać wspomniane trzy najlepsze detektory za zwycięzców jeśli chodzi o wyniki dla tej metryki. Dużym zaskoczeniem może wydawać się obecność drzewa decyzyjnego w czołówce najlepszych wyników, zwłaszcza na tak wysokim miejscu. W porównaniu do reszty detektorów w czołówce, cechuje się zdecydowanie najmniejszą złożonością. Las losowy, który uśrednia wyniki dla wielu drzew decyzyjnych wchodzących w skład tego detektora, zdołał osiągnąć wynik marginalnie lepszy od pojedynczego drzewa. Odnosząc się zaś do technik dla niezbalansowanych zbiorów danych, żadna z nich nie okazała się być pomocna. Lepiej poradziły sobie detektory bez wyrównywania braku balansu klas, patrząc na wyniki dla lasu losowego, czy też drzewa decyzyjnego.

5.1.1.2 Średnia precyzja

Bazowym wynikiem dla średniej precyzji będzie pole pod krzywą precyzji w funkcji czułości, która jest poziomą linią o wysokości równej liczbie transakcji związanych z oszustwem, w stosunku do całkowej liczby transakcji. Innymi słowy, będzie to odsetek oszustw w zbiorze testowym. W praktyce zatem, bazowa wartość średniej precyzji wynosi w przybliżeniu 0.007. Szczegółowe wyniki dla średniej precyzji znajdują się w tabeli 2.

Model	Źródło	Średnia precyzja	Technika dla niezbalansowanych zbiorów danych
gMLP	TSAI	0.715	Nie dotyczy
TSiT	TSAI	0.705	Nie dotyczy
LSTM	TSAI	0.700	Nie dotyczy
XGBoost	podręcznik	0.699	Brak
TST	TSAI	0.692	Nie dotyczy
Agregacja z DD	podręcznik	0.691	Brak
Las losowy	podręcznik	0.687	Brak
GRU	TSAI	0.683	Nie dotyczy
Agregacja z DD	podręcznik	0.682	RandUnder
InceptionTime	TSAI	0.672	Nie dotyczy

Tabela 2: Detektory i zastosowanie techniki dla niezbalansowanych zbiorów danych z najwyższym wynikiem dla średniej precyzji

Na pierwszy rzut oka nasuwa się to, że wśród najlepszych wyników na pierwszym miejscu znalazł się detektor gMLP, dla którego wynik jest znacząco odstaje od reszty. Natomiast sama czołówka najlepszych detektorów wygląda podobnie jak w przypadku wyniku F1, zmieniła się jedynie kolejność. Detektor gMLP, poprzednio zajmując trzecie miejsce, teraz znalazł się na pierwszym, LSTM z pierwszego miejsca spadł na trzecie, XGBoost drugie miejsce zamienił na czwarte. Detektor TSiT, który dla poprzedniej metryki zajął szóste miejsce, dla średniej precyzji osiągnął drugi najlepszy wynik. Odnosząc się do rozpiętości 10 najlepszych wyników dla średniej precyzji, jest ona porównywalna do rezultatów dla wyniku F1. Detektory na dalszych pozycjach w tabeli wyników znacząco osiągnęły gorszy rezultat dla tej metryki. Ostatnią kwestią jest w końcu obecność detektora, który wykorzystał technikę dla niezbalansowanych zbiorów danych, a mianowicie zmniejszenie liczebności klasy większościowej w sposób losowy. Daje się jednak zauważyć, że ten sam detektor, czyli agregacja z drzewami decyzyjnymi, ale bez użycia wspomnianej techniki redukowania niezbalansowania klas, osiągnął lepszy wynik.

5.1.1.3 Card Precision@100

Dla metryki CP@k, najwyższy osiągalny wynik, w przeciwieństwie do pozostałych metryk, może być niższy od 1. Dzieje się tak w przypadku, gdy liczba kart rzeczywiście powiązanych z oszustwem będzie niższa niż parametr k. Jest to szczególnie ważne ze względu na to, że autorzy podręcznika w przygotowanych przez siebie funkcjach, wprowadzili mechanizm, który usuwa karty rzeczywiście powiązane z oszustwem po

poprawnej detekcji. Dla konkretnego przykładu podają, że wynik Card Precision@100 równy 0.33 przekłada się na około 60% rozpoznanych przez detektor kart, dla których faktycznie wystąpiło oszustwo. Szczegółowe wyniki dla CP@100 znajdują się w tabeli 3.

Model	**Źródło**	**CP@100**	**Technika dla niezbalansowanych zbiorów danych**
Agregacja z DD	podręcznik	0.314	RandUnder
gMLP	TSAI	0.313	Nie dotyczy
TSiT	TSAI	0.307	Nie dotyczy
XCM	TSAI	0.307	Nie dotyczy
Drzewo decyzyjne	podręcznik	0.307	SMOTE
Las losowy	podręcznik	0.306	RandUnder
InceptionTime	TSAI	0.306	Nie dotyczy
TST	TSAI	0.304	Nie dotyczy
GRU	TSAI	0.304	Nie dotyczy
Drzewo decyzyjne	podręcznik	0.304	SMOTE + RandUnder

Tabela 3: Detektory i zastosowanie techniki dla niezbalansowanych zbiorów danych z najwyższym wynikiem dla CardPrecision@100

W związku ze specyfiką CP@k, trudniej jest wysunąć podobne wnioski jeśli chodzi o to, czy konkretne detektory odznaczają się od reszty pod względem osiągniętych wyników, czy też jak duża jest rozpiętość wyników wśród nich. Na pewno daje się zauważyć, że dla większej liczby detektorów z najlepszymi wynikami, użyto techniki dla niezbalansowanych zbiorów danych. Precyzując, wszystkie z detektorów przedstawionych oryginalnie w podręczniku wykorzystało jedną z dostępnych technik. Pojawia się tu użycie techniki SMOTE, zarówno jako jedynej techniki dla detektora, jak i w połączeniu ze zmniejszaniem liczebności klasy większościowej w sposób losowy. Ponownie, drzewo decyzyjne pojawiło się na liście, tym razem aż dwukrotnie, a w połączeniu z techniką SMOTE uzyskało trzeci najlepszy wynik, ex aequo z bardziej złożonymi detektorami TSiT i XCM. Natomiast detektorem, który osiągnął najwyższy wynik dla CP@100, okazał się ten wykorzystujący agregację z drzewami decyzyjnymi, a tuż za nim znalazł się detektor gMLP, osiągając marginalnie gorszy wynik.

5.1.1.4 AUC ROC

Spośród wszystkich metryk, dla których górna granica wyniku to 1, to właśnie dla AUC ROC można spodziewać się relatywnie najwyższych wartości. Powodem jest to, że dla ,,losowego'' klasyfikatora, który będzie przypisywał każdej transakcji prawdopodobieństwo bycia oszustwem z rozkładu jednolitego między 0 a 1, wynik AUC ROC będzie równy 0.5. Jak zostało wspomniane w rozdziale o metodach detekcji anomalii, AUC ROC nie jest niezawodną metryką dla niezbalansowanych zbiorów danych. Dlatego też, można potraktować tę metrykę jako mniej znaczącą od pozostałej trójki. Mimo to, wyniki dla AUC ROC również zostały zebrane i znajdują się w tabeli 4.

Dla ostatniej z omawianych metryk ewaluacyjnych, mających za zadanie porównać między sobą detektory, najlepszy wynik uzyskał las losowy. Jednak różnice pomiędzy

Model	Źródło	AUC ROC	Technika dla niezbalansowanych zbiorów danych
Las losowy	podręcznik	0.907	RandUnder
TSiT	TSAI	0.906	Nie dotyczy
gMLP	TSAI	0.905	Nie dotyczy
Agregacja z DD	podręcznik	0.904	RandUnder
GRU	TSAI	0.899	Nie dotyczy
Las losowy	podręcznik	0.899	Brak
LSTM	TSAI	0.898	Nie dotyczy
ResNet	TSAI	0.898	Nie dotyczy
MLSTM-FCN	TSAI	0.895	Nie dotyczy
TST	TSAI	0.894	Nie dotyczy

Tabela 4: Detektory i zastosowanie techniki dla niezbalansowanych zbiorów danych z najwyższym wynikiem dla AUC ROC

czterema najlepszymi detektorami są marginalne, dlatego też kolejność wspomnianego lasu losowego i następnych w czołówce detektorów TSiT, gMLP, i agregacji z drzewami losowymi mogłaby ulec zmianie, w wypadku drobnych zmian w zbiorze danych. Podobnie można by przeanalizować rozpiętość wyników dla AUC ROC, gdzie różnica między najlepszych wynikiem i tym na dziesiątym miejscu jest zdecydowanie mniejsza niż dla wyniku F1 czy średniej precyzji. Ostatnim spostrzeżeniem jest obecność tej samej techniki dla niezbalansowanych zbiorów danych, dla dwóch detektorów będących lasem losowym i agregacją z drzewami decyzyjnymi. Precyzując, zmniejszenie liczebności klasy większościowej w sposób losowy pomogło wspomnianym detektorom uzyskać wyższy wynik AUC ROC, niż gdyby żadna z technik dla niezbalansowanych zbiorów danych nie została użyta.

5.1.2 Czas treningu i wnioskowania

Detektory wykorzystujące uczenie głębokie, ze względu na większą liczbę parametrów do optymalizacji, potrzebują więcej czasu na uczenie niż klasyczne modele uczenia maszynowego. To samo dotyczy czasu potrzebnego, aby ocenić ostateczny model za pomocą zbioru testowego, czyli tzw. wnioskowania (ang. inference). Jest to jeden z największych minusów modeli należących do tej grupy. Dlatego też porównywanie czasów pomiędzy modelami klasycznymi i modelami uczenia głębokiego nie przyniesie niespodziewanych rezultatów. Niemniej, porównanie czasów wewnątrz grupy ma większy sens. Z racji, że modele klasyczne i ich czasy zostały dokładnie omówione przez twórców podręcznika, w tej podsekcji uwaga poświęcona zostanie modelom wykorzystującym uczenie głębokie. Mając na uwadze wpływ rozmiaru grupy danych na czas uczenia i wnioskowania, porównane zostaną ze sobą detektory dla stałego rozmiary grupy danych równego 64. Liczba epok, która również decyduje o czasie uczenia i wnioskowania, dla każdego z przedstawionych poniżej detektorów, została wybrana przy pomocy techniki wczesnego zatrzymywania z cierpliwością o wartości 2. Współczynnik uczenia również został ustawiony na stałą wartość wynoszącą 1e-4. Tyczy się to również detektorów wykorzystujących bibliotekę TSAI. Na potrzeby tego testu zrezygnowano z automatycznego wybierania optymalnej wartości współczynnika uczenia i stosowania polityki

jednego cyklu jako algorytmu-planisty. Jeśli chodzi o resztę hiperparametrów, specyficznych dla konkretnej architektury detektora, pozostawiono je bez zmian w stosunku do domyślnych wartości. Przez zastosowanie wczesnego zatrzymywania, liczba epok między detektorami będzie się różnić. Dlatego też, oprócz porównywania całkowitego czasu treningu, porównano czas treningu na epokę. Szczegółowe wyniki znajdują się w tabeli 5.

Model	Źródło modelu	Czas [s]		
		treningu	wnioskowania	treningu na epokę
LSTM	TSAI	46.93	1.342	11.733
FCN	TSAI	66.648	1.825	16.662
XCM	TSAI	75.989	2.432	18.997
LSTM-FCN	TSAI	78.984	2.476	19.746
GRU-FCN	TSAI	93.65	2.167	18.730
GRU	TSAI	100.777	1.201	11.197
ResCNN	TSAI	115.721	2.706	23.144
MLSTM-FCN	TSAI	120.909	3.314	24.182
ResNet	TSAI	125.659	4.401	31.415
OmniScaleCNN	TSAI	181.951	4.581	30.325
InceptionTime	TSAI	249.840	7.824	49.968
TSiT	TSAI	252.782	12.431	63.196
gMLP	TSAI	335.773	5.895	41.972
TST	TSAI	588.483	6.625	36.780
MLP	podręcznik	98.318	0.005	3.933
Autokoder	podręcznik	199.102	1.019	4.856
Autokoder + MLP	podręcznik	354.266	0.818	8.239
LSTM z atencją	podręcznik	470.185	10.544	27.658
CNN	podręcznik	595.455	10.043	24.810
LSTM	podręcznik	709.893	10.988	25.353

Tabela 5: Wyniki dla testu czasu treningu i wnioskowania dla detektorów wykorzystujących uczenie głębokie

Najszybszym detektorem, zarówno jeśli chodzi o czas treningu i wnioskowania, okazała się być implementacja architektury MLP, stworzona przez autorów podręcznika. Jest to jednocześnie najprostsza sieć, która składa się jedynie z jednej warstwy ukrytej o rozmiarze 1000. Skupiając się jeszcze na detektorach z podręcznika, zarówno autokoder jak i autokoder połączony z architekturą MLP, pozwalają na szybszy trening i wnioskowanie w porównaniu do przetestowanych detektorów, bazujących na bibliotece TSAI. Jeśli chodzi o detektory TSAI, zauważalnie najszybszymi okazały się być, w następującej kolejności, sieć GRU oraz wspomniana już sieć LSTM. Na drugim końcu znajdują się nowsze, bardziej złożone architektury detektorów. Najwolniejszymi detektorami, w kolejności największych czasów treningu na epokę, są TSiT, InceptionTime, gMLP i TST.

5.1.3 Optymalne hiperparametry

Jak już wcześniej wspomniano, detektory oparte o uczenie głębokie dzielą między sobą niektóre z hiperparametrów. Wśród detektorów uczenia głębokiego, można wy-

różnić powszechne hiperparametry jak liczba epok, współczynnik uczenia się, rodzaj optymalizatora czy rozmiar grupy danych. Z racji, że algorytm Adam jest optymalizatorem, który został wybrany dla każdego z testowanych detektorów, w tej podsekcji zostaną ze sobą porównane pozostałe. Przedstawione zostaną wartości wspomnianych hiperparametrów, dla których dany detektor uzyskał najlepszy wynik podczas wyszukiwania tych najbardziej optymalnych. Dla detektorów wykorzystujących bibliotekę TSAI, jako wartość współczynnika uczenia, podano maksymalną jego wartość, która jest osiągana podczas treningu z wykorzystaniem polityki jednego cyklu. Szczegółowe informacje zostały przedstawione w tabeli 6. Dodatkowo, w ostatniej kolumnie tabeli, wyświetlono całkowitą liczbę trenowalnych parametrów detektorów. Trenowalnymi parametrami określa się te parametry detektora, które są aktualizowane w trakcie uczenia. Dla uproszczenia, w tej podsekcji będą po prostu określane jako parametry.

Model	Źródło modelu	Rozmiar grupy danych	Współczynnik uczenia	Liczba epok	Liczba tren. parametrów
Autokoder	podręcznik	64	1e-4	49	7 235
MLP	podręcznik	64	1e-3	20	8 501
Autokoder + MLP	podręcznik	64, 64	1e-4, 1e-4	49, 90	9 036
CNN	podręcznik	64	1e-3	10	74 201
LSTM	podręcznik	128	1e-3	5	97 801
LSTM z atencją	podręcznik	128	1e-4	10	119 501
TST	TSAI	256	3.6e-3	10	21 729
XCM	TSAI	64	7.6-3	8	22 019
TSiT	TSAI	64	2.1e-3	5	33 153
FCN	TSAI	256	2.5e-3	7	72 833
GRU	TSAI	64	3.6e-3	5	95 801
MLSTM-FCN	TSAI	128	3e-3	5	122 773
LSTM	TSAI	128	2.1e-3	10	127 701
GRU-FCN	TSAI	64	3e-3	5	163 713
LSTM-FCN	TSAI	64	2.5e-3	5	194 613
ResCNN	TSAI	128	3.6e-3	9	263 298
ResNet	TSAI	64	2.5e-3	7	485 505
gMLP	TSAI	64	2e-4	12	700 371
InceptionTime	TSAI	128	2.1e-3	5	1 820 545
OmniScaleCNN	TSAI	128	1.2e-3	8	10 610 756

Tabela 6: Zestawienie optymalnych hiperparametrów znalezionych w trakcie wyszukiwania, z naciskiem na hiperparametry typowe pośród modeli uczenia głębokiego

Rozmiar grupy danych, powyżej pewnego progu liczby parametrów w okolicach 75-95 tysięcy, przestaje osiągać wartości większe niż 128. O współczynniku uczenia trudno powiedzieć, że ma tendencję osiągać konkretne rzędy wartości dla danej grupy detektorów. Małe wartości współczynnika uczenia, rzędu 1e-4, były wybierane przez autorów podręcznika podczas uczenia detektorów opartych o architekturę autokodera, a jednocześnie zostały wybierane podczas wyszukiwania hiperparametrów dla detektorów o liczbie parametrów równej 700 tysięcy. Mimo to, dla przeważającej liczby detektorów, współczynnik uczenia osiąga wartości rzędu 1e-3. Widać zależność między współczynni-

kiem uczenia, a liczbą epok − współczynnikom uczenia o wartości rzędu 1e-4 odpowiada większa, patrząc na średnią liczbę dla testowanych tu detektorów, liczba epok. Liczba parametrów jest zdecydowanie najbardziej zróżnicowaną właściwością wśród detektorów, Na szczególną uwagę zasługują dwa najbardziej złożone, pod względem liczby parametrów, detektory, czyli OmniScaleCNN i InceptionTime. Pod względem liczby parametrów różnią się one od reszty detektorów o kolejno dwa i jeden rzędy wielkości.

5.2 Wyjaśnianie anomalii

W tej sekcji pracy przedstawione zostaną wyniki eksperymentów dla metod wyjaśniania TS COIN i SHAP.

5.2.1 Ewaluacja wyniku odstawania

W celu ewaluacji wyniku odstawania zastosowana została ta sama metoda, którą zaproponowali autorzy w artykule COIN. Metoda ta wykorzystuje faktyczne etykiety, aby sprawdzić, czy pod-sekwencje błędnie zaklasyfikowane przez detektor mają średnio niższy wynik wartości odstających niż pod-sekwencje zaklasyfikowane poprawnie przez detektor. W przypadku większości detektorów średni wynik wartości odstających okazuje się wyższy dla prawidłowych predykcji niż dla błędnych. Ogólnie rzecz biorąc, wynik wartości odstającej działa lepiej w przypadku modeli TSAI, tj. różnica między średnim wynikiem wartości odstającej dla prawidłowych predykcji, a średnim wynikiem wartości odstającej dla błędnych prognoz jest wyższa niż w przypadku modeli podręcznikowych. Szczegółowe wyniki ewaluacji znajdują się w tabeli 7. Przedstawiona została tam proporcja średniego wyniku odstawania dla przykładów prawdziwie pozytywnych *PP* i średniego wyniku odstawania dla przykładów fałszywie pozytywnych *FP*. Im bardziej wynik proporcji jest większy od 1, tym wynik odstawania lepiej oddziela faktyczne anomalie od fałszywych alarmów. We wspomnianej tabeli, jak i w kolejnych tabelach w tej sekcji, oznaczenie *HT* na końcu nazwy modelu wskazuje, że przeprowadzono dla niego proces przeszukiwania hiperparametrów. Modele bez oznaczenia *HT* zostały wytrenowane z domyślnymi hiperparametrami.

Średni wynik wartości odstających obliczony dla pod-sekwencji zaklasyfikowanych jako odstające, w porównaniu do średniego wyniku wartości odstających obliczony dla próby pod-sekwencji zaklasyfikowanych jako typowe, jest zauważalnie wyższy. Wyniki dla wszystkich testowanych detektorów są zgodne z tą regułą. Szczegółowe wyniki znajdują się w tabeli 8.

5.2.2 SHAP jako alternatywna metoda wyjaśniania anomalii

SHAP został wybrany jako konkurencyjna metoda wyjaśniania predykcji detektora. Do przybliżenia wartości SHAP dla testowanych detektorów wybrano algorytm DeepExplainer. Jest to jedyny algorytm z biblioteki SHAP, który jest w stanie generować wartości na podstawie sekwencyjnego zbioru danych i sekwencyjnego detektora. W przypadku tego typu zbiorów danych i detektorów, SHAP okazał się nie być metodą niezależną od modelu. Więcej o wyzwaniach związanych z testowaniem algorytmu DeepExplainer można przeczytać w rozdziale 6.

Model	Źródło modelu	$\frac{PP}{FP}$ dla średniego wyniku odstawania
TSiT	TSAI	1.444
GRU	TSAI	1.385
LSTM	TSAI	1.353
XCM	TSAI	1.345
InceptionTime	TSAI	1.248
TST	TSAI	1.178
OmniScaleCNN	TSAI	1.156
ResCNN	TSAI	1.153
GRU-FCN	TSAI	1.135
LSTM z atencją	podręcznik	1.128
CNN HT	podręcznik	1.120
gMLP	TSAI	1.109
MLSTM-FCN	TSAI	1.080
LSTM z atencją HT	podręcznik	1.080
LSTM	podręcznik	1.020
LSTM-FCN	TSAI	1.020
FCN	TSAI	1.009
LSTM HT	podręcznik	1.005
ResNet	TSAI	0.972
CNN	podręcznik	0.963

Tabela 7: Proporcja średniego wyniku odstawania dla przykładów prawdziwie pozytywnych i średniego wyniku odstawania dla przykładów fałszywie pozytywnych

5.2.3 NDCG

Zarówno SHAP, jak i TS COIN, dla każdej cechy wejściowej, zwracają liczbę określającą wpływ danej cechy na wynik predykcji. Biorąc pod uwagę brak możliwości bezpośredniego porównania wartości z anormalnego wyniku atrybutu i wartości SHAP, a także fakt, że wartości SHAP dla detektorów z podręcznika różnią się od tych utworzonych za pomocą TSAI, jako jedną z metod porównywania skuteczności TS COIN i SHAP wybrano porównanie rankingów najbardziej wpływowych cech. W przypadku SHAP, ranking najbardziej wpływowych cech jest obliczany przez uśrednienie bezwzględnych wartości SHAP dla danej funkcji. To samo można zrobić w przypadku anormalnego wyniku atrybutu – jedyna różnica polega na tym, że dla anormalnego wynik atrybutu nie ma potrzeby odnoszenia się do bezwzględnej wartości, ponieważ z definicji wartości nie są niższe niż 0.

Jako miernik jakości rankingu wybieramy Normalized Discounted Cumulative Gain, w skrócie NDCG, [34]. NDCG można zdefiniować jako:

$$NDCG = \frac{DCG}{IDCG}, \tag{5.1}$$

gdzie DCG to Discounted Cumulative Gain, a IDCG to Ideal Discounted Cumulative Gain. W przypadku wyjaśniania predykcji testowanych detektorów, DCG mierzy łączny wpływ cech wejściowych na podstawie ich pozycji w rankingu. Gain (ang. zysk) lub wpływ oblicza się, dodając wynik istotności (ang. relevance score) każdej cechy od góry

Model	Źródło modelu	Średni wynik odstawania dla elementów	
		odstających	typowych
CNN	podręcznik	5.736	1.135
LSTM	podręcznik	5.402	1.130
LSTM Attention	podręcznik	5.389	1.126
CNN HT	podręcznik	4.617	1.085
LSTM HT	podręcznik	5.552	1.128
LSTM Attention HT	podręcznik	5.769	1.129
LSTM	TSAI	5.422	1.136
FCN	TSAI	5.950	1.099
gMLP	TSAI	5.234	1.136
GRU-FCN	TSAI	5.879	1.123
GRU	TSAI	5.118	1.133
InceptionTime	TSAI	5.392	1.104
LSTM-FCN	TSAI	5.800	1.132
MLSTM-FCN	TSAI	5.722	1.108
OmniScaleCNN	TSAI	5.026	1.103
ResCNN	TSAI	5.856	1.128
ResNet	TSAI	5.868	1.107
TSiT	TSAI	4.682	1.127
TST	TSAI	5.396	1.136
XCM	TSAI	5.368	1.130

Tabela 8: Porównanie średniego wyniku odstawania dla elementów odstających ze średnim wynikiem odstawania dla próbki elementów typowych

do dołu listy rankingowej, przy czym każdy wynik jest ,,przeceniany'' wraz ze spadkiem rangi. DCG na określonej pozycji w rankingu p definiuje się jako:

$$DCG_p = rel_1 + \sum_{i=2}^{p} \frac{rel_i}{log_2(i+1)}, \tag{5.2}$$

gdzie rel_i reprezentuje odpowiednio ,,przecenioną'' istotność na pozycji i. IDCG mierzy DCG dla idealnego porządku, lub w przypadku tej pracy dla, zgodnej z prawdą, ustalonej istotności cech wejściowych:

$$IDCG_p = \sum_{i=1}^{|REL_p|} \frac{rel_i}{log_2(i+1)}, \tag{5.3}$$

gdzie REL_p reprezentuje listę cech uporządkowanych według ich wpływu na predykcję, aż do pozycji w rankingu p.

Faktyczna kolejność cech wejściowych w rankingu tych najbardziej wpływowych opiera się na informacjach o sposobie wygenerowania zestawu danych. Na 7 z 15 wartości cech wejściowych mają bezpośredni wpływ procedury generowania scenariuszy oszustw stosowane do tworzenia zbioru danych. Uwzględniana jest również liczba transakcji zmodyfikowanych przez każdy ze scenariuszy. Na tej podstawie przygotowywany jest ranking 7 najbardziej wpływowych cech. Dodatkowo obliczany jest współczynnik korelacji Tau-Kendall, którego to wartości potwierdzają kolejność tych 7 najbardziej wpływowych cech w rankingu i proponują kolejność w rankingu dla pozostałych 8. Z tego

powodu NDCG jest obliczany dla wszystkich 15 cech wejściowych, a także uwzględniając tylko 7 najlepszych cech w rankingu. Rozmiar danych tła, używanych podczas obliczania wartości SHAP, jest ustawiony na 1000. Ze względu na to, że utworzony faktyczny ranking najbardziej wpływowych cech nie wyróżnia cech w czasie, tj. nie szereguje cech dla każdego kroku pod-sekwencji z osobna, obliczone wartości anormalnego wyniku atrybutu oraz wartości SHAP są sumowane dla wszystkich kroków pod-sekwencji, dla danej cechy wejściowej. Szczegółowy wynik testu NDCG przedstawiony jest w tabeli 9.

Uśredniając wyniki dla wszystkich detektorów, średni NDCG dla anormalnego wyniku atrybutu, biorąc pod uwagę wszystkie cechy wejściowe, wynosi 0.898. Jeśli weźmiemy pod uwagę tylko ranking 7 najbardziej wpływowych cech, średni NDCG dla anormalnego wyniku atrybutu wynosi 0.750. Wyniki NDCG przy wykorzystaniu wartości z SHAP, zarówno dla wszystkich cech wejściowych, jak i biorąc pod uwagę tylko 7 najlepszych cech, wynoszą odpowiednio 0.793 i 0.584.

Model	Źródło modelu	AWA NDCG		SHAP NDCG	
		15 cech	7 cech	15 cech	7 cech
CNN	podręcznik	0.923	0.801	0.785	0.633
LSTM	podręcznik	0.916	0.802	0.815	0.5602
LSTM z atencją	podręcznik	0.740	0.486	0.794	0.622
CNN HT	podręcznik	0.890	0.747	0.786	0.621
LSTM HT	podręcznik	0.928	0.805	0.933	0.801
LSTM z atencją HT	podręcznik	0.935	0.798	0.725	0.478
LSTM	TSAI	0.915	0.764	0.801	0.575
FCN	TSAI	0.941	0.828	0.717	0.418
gMLP	TSAI	0.933	0.770	0.731	0.419
GRU-FCN	TSAI	0.932	0.805	0.894	0.790
GRU	TSAI	0.824	0.624	0.705	0.355
InceptionTime	TSAI	0.800	0.638	0.864	0.603
LSTM-FCN	TSAI	0.965	0.891	0.773	0.647
MLSTM-FCN	TSAI	0.944	0.866	0.758	0.598
OmniScaleCNN	TSAI	0.860	0.661	0.736	0.397
ResCNN	TSAI	0.940	0.793	0.812	0.666
ResNet	TSAI	0.930	0.805	0.866	0.666
TSiT	TSAI	0.865	0.644	0.745	0.555
TST	TSAI	0.948	0.838	0.822	0.716
XCM	TSAI	0.827	0.624	0.803	0.569

Tabela 9: NDCG dla AWA (anormalny wynik atrybutu) z metody TS COIN i dla biblioteki SHAP

5.2.4 Stabilność

Jako inny sposób porównania TS COIN i SHAP wybrano stabilność [50], czyli właściwość metod wyjaśniania, która mierzy jak podobne są wyjaśnienia dla podobnych elementów, w ramach konkretnego detektora. W przypadku tej pracy, będzie mierzyć jako podobne są wyjaśnienia dla podobnych do siebie transakcji. Do obliczenia stabilności wykorzystany został współczynnik Lipschitza [7]. Współczynnikiem Lipschitza określa

się największym stosunkiem zmiany na wyjściu do zmiany na wejściu funkcji f. Jest on zdefiniowany jako najmniejsza stała $L > 0$ w w warunku Lipschitza. Warunek Lipschitza definiuje się następująco;

$$\|f(x) - f(y)\| \leqslant L\|x - y\| \text{ dla wszystkich } x, y \in \mathcal{X}. \tag{5.4}$$

Tak jak w przypadku NDCG, rozmiar danych tła dla algorytmu DeepExplainer jest ustawiony na 1000. Szczegółowy wynik testu stabilności przedstawiony jest w tabeli 10.

Model	Źródło modelu	AWA stabilność		SHAP stabilność	
		Średnia	σ	Średnia	σ
CNN	podręcznik	0.977	0.023	0.962	0.040
LSTM	podręcznik	0.975	0.023	0.928	0.074
LSTM Attention	podręcznik	0.972	0.025	0.946	0.051
CNN HT	podręcznik	0.976	0.026	0.954	0.051
LSTM HT	podręcznik	0.973	0.024	0.951	0.048
LSTM Attention HT	podręcznik	0.974	0.024	0.959	0.040
LSTM	TSAI	0.975	0.024	0.831	0.160
FCN	TSAI	0.977	0.025	0.982	0.020
gMLP	TSAI	0.971	0.026	0.818	0.150
GRU-FCN	TSAI	0.975	0.024	0.956	0.040
GRU	TSAI	0.973	0.024	0.800	0.173
InceptionTime	TSAI	0.978	0.024	0.584	0.399
LSTM-FCN	TSAI	0.977	0.023	0.971	0.026
MLSTM-FCN	TSAI	0.977	0.024	0.960	0.043
OmniScaleCNN	TSAI	0.977	0.024	0.875	0.126
ResCNN	TSAI	0.973	0.023	0.958	0.038
ResNet	TSAI	0.976	0.026	0.970	0.030
TSiT	TSAI	0.970	0.025	0.891	0.086
TST	TSAI	0.970	0.025	0.860	0.118
XCM	TSAI	0.975	0.023	0.925	0.070

Tabela 10: Średnia stabilność i jej odchylenie standardowe (oznaczone jako σ) dla AWA (anormalny wynik atrybutu) z metody TS COIN i dla biblioteki SHAP

Uśredniając wyniki dla wszystkich detektorów, średnia stabilność dla anormalnego wyniku atrybutu wyniosła 0.975, a dla SHAP 0.904. Ponownie, w przypadku uśrednienia wyników dla wszystkich detektorów, średnie odchylenie standardowe dla anormalnego wyniku atrybutu wyniosło 0.024, a dla SHAP 0.089.

5.2.5 Czas wykonywania

W ramach ostatniego testu zmierzono czas wykonywania obu metod wyjaśniania. Zaczynając od TS COIN − czasy wykonywania tej metody dla wszystkich detektorów mieszczą się w przedziale od 12 minut do 23 minut, a średni czas wyniósł 18 minut. Warto przypomnieć, że użyta implementacja TS COIN wykorzystuje bibliotekę do obliczeń równoległych. Szczegółowy wynik testu czasu wykonywania przedstawiony jest w tabeli 11.

Model	Źródło modelu	Czas wykonywania [s]
ResNet	TSAI	753.43
OmniScaleCNN	TSAI	761.34
MLSTM-FCN	TSAI	786.17
XCM	TSAI	888.43
ResCNN	TSAI	950.85
TSiT	TSAI	954.52
LSTM-FCN	TSAI	958.44
TST	TSAI	986.78
FCN	TSAI	1038.47
CNN HT	podręcznik	1069.26
InceptionTime	TSAI	1072.29
GRU	TSAI	1166.63
LSTM	TSAI	1191.66
GRU-FCN	TSAI	1196.04
LSTM	podręcznik	1242.33
LSTM Attention HT	podręcznik	1254.66
LSTM HT	podręcznik	1302.46
LSTM Attention	podręcznik	1308.19
gMLP	TSAI	1351.55
CNN	podręcznik	1370.18

Tabela 11: Czas wykonywania metody TS COIN dla wszystkich wykrytych anomalii przez dany detektor

Czas wykonywania dla SHAP zależy przede wszystkim od wybranego rozmiaru danych tła. Należy jednak pamiętać, że im mniejszy rozmiar danych tła, tym mniej precyzyjne będą wygenerowane wartości SHAP. Dlatego zdecydowano się zmierzyć czas wykonywania SHAP, wykorzystując największy możliwy rozmiar danych tła, czyli wielkość całego zbioru treningowego. Pomiar czasu wykonywania SHAP dla przedstawionych detektorów i danych sekwencyjnych stanowił większe wyzwanie, w porównaniu do pomiarów dla TS COIN. Istotny wpływ na wynik próby wygenerowania wartości SHAP i zmierzone czasy ma wybór jednostki do wykonywania obliczeń, a mianowicie wybór między CPU i GPU. W przypadku niektórych detektorów, przy próbie wykonywania obliczeń na GPU, SHAP zwraca błąd związany z niekompatybilnością danej implementacji architektury detektora i algorytmu DeepExplainer, nawet dla najmniejszych rozmiarów danych tła. Z drugiej strony, przy próbie wygenerowania wartości SHAP z wykorzystaniem całego zbioru treningowego jako danych w tle, wszystkie testowane detektory zwracają błąd informujący o przekroczeniu dostępnej pamięci (ang. Out of Memory, w skrócie OOM) użytego GPU. Oprócz danych tła o rozmiarze równym rozmiarowi całego zbioru treningowego, zmierzony został również czas wykonywania dla rozmiarów 100, 500, 1000, 2000 i 5000. Szczegółowy wynik testu czasu wykonywania SHAP z wykorzystaniem GPU przedstawiony jest w tabeli 12.

Podczas wykonywania obliczeń na GPU, czasy wykonywania dla tych detektorów, które nie napotkały problemów przy obliczaniu wartości SHAP, skalują się liniowo wraz ze wzrostem rozmiaru danych tła. Nie dotyczy to pomiarów z maksymalnym rozmiarem danych tła, które zwracają błąd OOM, odpowiadający za brak dostępnej pamięci

Model	Źródło modelu	Czas wykonywania [s] z rozmiarem danych tła					
		100	500	1000	2000	5000	Maks.
LSTM	podręcznik	2.33	7.2	13.8	28.3	67.97	OOM
LSTM Attention	podręcznik	2.5	7.21	13.72	27.14	65.55	OOM
LSTM Attention HT	podręcznik	2.19	6.74	13.06	24.74	61.66	OOM
LSTM HT	podręcznik	1.92	8.15	14.99	31.86	75.78	OOM
FCN	TSAI	2.36	7.48	13.55	27.38	72.13	OOM
gMLP	TSAI	5.63	14.43	28.36	55.11	158.64	OOM
GRU-FCN	TSAI	3.15	8.95	16.74	35.03	86.64	OOM
GRU	TSAI	2.07	10.32	19.86	39.82	91.37	OOM
LSTM-FCN	TSAI	3.73	10.48	19.78	44.5	114.01	OOM
LSTM	TSAI	1.92	9.66	19.61	39.17	92.27	OOM
MLSTM-FCN	TSAI	4.28	10.16	17.75	35.78	99.47	OOM
OmniScaleCNN	TSAI	5.07	17.92	39.05	95.3	OOM	OOM
ResCNN	TSAI	5.31	13.05	23.05	45.49	107.66	OOM
ResNet	TSAI	5.06	10.2	18.11	35.72	93.95	OOM
TSiT	TSAI	10.36	22.15	40.62	83.73	202.32	OOM
TST	TSAI	7.75	17.98	33.75	67.85	212.16	OOM
XCM	TSAI	4.78	13.3	24.31	48.25	114.57	OOM

Tabela 12: Czas wykonywania SHAP z wykorzystaniem GPU

GPU. W przypadku obliczeń z wykorzystaniem CPU, SHAP jest niekompatybilny z większą liczbą detektorów niż podczas korzystania z GPU, niezależnie od ustawionego rozmiaru danych tła. Przewaga CPU nad GPU polega na tym, że obliczenia na CPU, ze względu na brak błędów związanych z brakiem pamięci, mogą być z powodzeniem wykonywane przy użyciu całego zestawu treningowego jako danych tła. Zajmuje to jednak znacznie więcej czasu – z tego powodu przetestowany został tylko jeden model z podręcznika i jeden model korzystający z TSAI. Dla pozostałych detektorów test czasu wykonywania nie został przeprowadzony na maksymalnym rozmiarze danych tła, jednak najprawdopodobniej dla większości z tych detektorów czasy wykonywania byłyby porównywalnie duże. Wniosek ten bazuje na obserwacji, że dla większości detektorów z pomyślnie przeprowadzonym pomiarem czasu wykonywania, czas obliczeń na CPU skaluje się wykładniczo w porównaniu z rozmiarem danych tła lub jest już wystarczająco wysoki, aby rozmiar danych tła wynoszący 5000, dorównywał lub przekraczał czas TS COIN dla odpowiadającego detektora. Szczegółowy wynik testu czasu wykonywania SHAP z wykorzystaniem CPU przedstawiony jest w tabeli 13.

Model	Źródło modelu	Czas wykonywania [s] z rozmiarem danych tła					
		100	500	1000	2000	5000	Maks.
CNN	podr.	1.19	7.07	20.48	67.11	330.58	54496.36
CNN HT	podr.	1.49	8.35	19.52	55.08	242.65	
LSTM	podr.	3.52	13.77	32.9	97.81	423.8	
LSTM Attention	podr.	4.57	17.24	39.49	114.44	473.74	
LSTM Attention HT	podr.	4.47	17.23	39.68	112.12	461.1	
LSTM HT	podr.	3.78	16.0	36.99	106.55	459.9	
FCN	TSAI	3.82	błąd	błąd	błąd	błąd	
gMLP	TSAI	55.86	252.36	508.06	1087.07	2808.41	
GRU-FCN	TSAI	11.33	błąd	błąd	błąd	błąd	
GRU	TSAI	3.62	13.09	33.17	93.45	391.73	
InceptionTime	TSAI	16.57	błąd	błąd	błąd	błąd	
LSTM-FCN	TSAI	14.59	błąd	błąd	błąd	błąd	
LSTM	TSAI	3.89	15.73	36.72	103.82	413.93	53367.26
MLSTM-FCN	TSAI	13.46	błąd	błąd	błąd	błąd	
OmniScaleCNN	TSAI	69.99	322.12	640.07	1350.87	3653.64	
ResCNN	TSAI	8.37	błąd	błąd	błąd	błąd	
ResNet	TSAI	8.95	błąd	błąd	błąd	błąd	
TSiT	TSAI	26.02	119.17	234.21	477.96	1342.63	
TST	TSAI	29.22	138.63	274.33	563.13	1581.79	
XCM	TSAI	10.71	50.28	106.88	240.92	823.18	

Tabela 13: Czas wykonywania SHAP z wykorzystaniem CPU

Rozdział 6
Dyskusja

6.1 Metody detekcji

6.1.1 Metryki ewaluacyjne

Pierwszą obserwacją, którą można wysunąć po analizie wyników metryk ewaluacyjnych dla metod detekcji, jest wyłączna obecność detektorów opartych w pełni o uczenie nadzorowane. Wśród detektorów i technik z najlepszymi wynikami zabrakło miejsca zarówno na modele wykorzystujące uczenie nienadzorowane i uczenie częściowo nadzorowane. Drugą obserwacją jest potwierdzenie wstępnych założeń, które opisywały problem z wybraniem jednoznacznie najlepszego detektora dla większej ilości metryk ewaluacyjnych. Dla każdej z nich, na pierwszym miejscu znalazł się inny detektor.

Patrząc jednak na czołówkę wyników wśród metryk, detektor oparty o architekturę gMLP wydaje się być tym najlepszym. Dla każdej z nich, detektor gMLP osiągnął jeden z najlepszych trzech wyników, co nie udało się żadnemu innemu detektorowi. Odczytując otrzymane wartości dla przetestowanych metryk, można zauważyć, że detektorowi gMLP brakowało na prawdę niewiele, aby zająć wszędzie pierwsze miejsce. W przypadku wyniku F1 i AUC ROC, od pierwszego miejsca dzieliła go tylko wartość 0.002, a żeby zajęć pierwsze miejsce dla metryki średnia precyzja, zabrakło mu jedynie 0.001. Na uwagę zasługuje też detektor TSiT, któremu udało się zając dwa razy drugi najlepszy wynik, i raz być na trzecim miejscu. Jedynie dla wyniku F1 zajął dalsze, ale wciąż wysokie, szóste miejsce. Jeśli zaś chodzi o pozostałe detektory i zastosowane wraz z nimi techniki dla niezbalansowanych zbiorów danych, można podzielić je na lepsze i gorsze. Dla każdej z metryk, w czołówce najlepszych wyników, widnieją takie detektory jak las losowy, TST, TSiT, GRU i wspomniany już gMLP. Niemałym zaskoczeniem może okazać się nieobecność detektora XGBoost wśród dziesięciu najlepszych wyników aż dla dwóch metryk, AUC ROC i Card Precision@100. Zwłaszcza, że wyniki dla innych detektorów, wykorzystujących uczenie zespołowe z wykorzystaniem drzew decyzyjnych, znalazły się w czołówce dla większej liczby metryk. Z drugiej strony, jeśli wybralibyśmy średnią precyzję i wynik F1 jako ważniejsze metryki ewaluacyjne, wówczas XGBoost okazałby się najlepszy z pośród wszystkich detektorów niewykorzystujących uczenia głębokiego. Dla wyniku F1 i średniej precyzji użycie technik dla niezbalansowanych zbiorów danych nie pomogło detektorom w osiągnięciu lepszych wyników. Odmienne wyniki otrzymano dla metryk CP@100 i AUC ROC − detektory z podręcznika, wykorzystujące jedną z technik dla niezbalansowanych zbiorów danych osiągnęły lepsze wyniki, niż bez ich wykorzystania.

6.1.2 Czas treningu i wnioskowania

Istotną obserwacją dla interpretacji wyników jest duża różnica, zarówno w czasie treningu na epokę jak i w czasie wnioskowania, pomiędzy implementacją architektury LSTM z podręcznika i z biblioteki TSAI. Obie implementacje architektury, w jedynej warstwie LSTM, miały domyślnie hiperparametr, odpowiedzialny za liczbę cech w stanie ukrytym h, równy 100. Jednym z czynników wpływających na taką dysproporcję jest fakt, że autorzy podręcznika, po warstwie LSTM, umieścili jeszcze jedną warstwę głęboką o rozmiarze 100. Spowodowało to zwiększenie liczby parametrów sieci z 46901 na 57001. Jednak jest to jedyna różnica jeśli chodzi o warstwy, z których składają się obydwie implementacje. Pozostałe hiperparametry również pozostają bez zmian. Oznacza to, że przynajmniej dla sieci LSTM, TSAI pozwala na szybszy trening i wnioskowanie. Zauważalne są także długie czasy wnioskowania dla detektorów sekwencyjnych z podręcznika, które ustępują pod tym względem tylko detektorowi TSiT. Szczególnie, gdy porówna się proporcję czasów wnioskowania do czasów treningu na epokę − wspomniane detektory sekwencyjne mają ją największą. Jeśli zaś chodzi o najdłuższe czasy treningu na epokę i wnioskowania wśród wszystkich testowanych detektorów wykorzystujących uczenie głębokie, to zmierzono je dla detektora TSiT. Sieć TSiT, która była jedną z najlepszych dla testowanych metryk ewaluacyjnych, potrzebuje ponad minutę dla jednej epoki treningu oraz ponad 12 sekund, aby wykonać wnioskowanie. Natomiast detektor gMLP, który wypadł najlepiej w poprzednim teście, potrzebuje ponad 40 sekund na trening dla jednej epoki, i prawie 6 sekund dla wnioskowania, co czyni go jednym z wolniejszych detektorów.

6.1.3 Optymalne hiperparametry

Detektory TST i TSiT, które wykorzystują architekturę Transformers [69], mimo swojej dużej (z definicji) złożoności i skomplikowanych mechanizmów, najlepsze wyniki osiągnęły dla małej liczby parametrów. Przyglądając się wybranym wartościom hiperparametrów podczas przeszukiwania, dla detektora TST domyślna liczba powtórzeń górnej warstwy kodera architektury Transformers, która domyślnie w implementacji wynosi trzy, została zredukowana do jednej. Zmniejszona została również liczba kwerend Q, kluczy K i wartości V (ang. queries, keys, values) w mechanizmie atencji, używanym w tych architekturach, z domyślnej wartości osiem, na wartość dwa. Spowodowało to prawie dwudziestokrotny spadek liczby parametrów detektora TST, w porównaniu do stanu, gdyby został wytrenowany z domyślnymi hiperparametrami. Dla detektora TSiT także, w procesie wyszukiwania hiperparametrów, wybrano mniejszą liczbę powtórzeń górnej warstwy kodera oraz mniejszą liczbę kwerend Q, kluczy K i wartości V, co w rezultacie również przyczyniło się do spadku liczby parametrów prawie dwudziestokrotnie. Można zatem stwierdzić, że obydwa detektory, w swojej domyślnej postaci, są zbyt złożone na potrzeby postanowionego problemu detekcji, dal użytego zbioru danych.

Niewspomnianym wcześniej sposobem na podział detektorów jest możliwość umiejscowienia ich w dwóch grupach − tej, która będzie obejmowała detektory, które w swojej architekturze wykorzystują warstwy konwolucyjne i tej, gdzie w architekturze detektora warstw konwolucyjnych nie ma. Dla dwudziestu przetestowanych detektorów wykorzystujących uczenie głębokie, dziesięć z nich wykorzystuje w swojej architekturze warstwy konwolucyjne. Bazując na tym podziale i zmierzonych liczbach parametrów okazuje się, że wśród sześciu najbardziej złożonych, pod względem liczby parametrów, detektorów aż pięć z nich posiada warstwy konwolucyjne.

6.1.4 Najlepsza metoda detekcji

Jeśli zależałoby nam na detektorze, który będzie uniwersalny w kwestii osiągania najlepszych wyników pośród wszystkich przedstawionych metryk ewaluacyjnych, byłby to detektor wykorzystujący architekturę gMLP. Jako, że w tej pracy głównym przypadkiem użycia jest wykorzystanie detektora jako kolejnej warstwy systemu wykrywania oszustw w transakcjach z użyciem kart kredytowych, należałoby się również pochylić nad wynikami z podsekcji o czasach treningu i wnioskowania. Tak, jak czas uczenia nie wydaje się być kluczowym czynnikiem przy wyborze najodpowiedniejszego detektora, tak już czas wnioskowania powinien spełniać określone wytyczne. Zgodnie z założeniami, detektor działający w takim systemie powinien analizować transakcje w czasie bliskim rzeczywistemu (ang. near-time), to znaczy predykcje nie muszą być generowane natychmiastowo, ale nie powinny przekraczać też ustalonego progu. Należałoby zatem wnikliwie przeanalizować, czy detektor gMLP, który potrzebował blisko 6 sekund na analizę prawie 60 tysięcy transakcji, spełnia wymagania działania w czasie bliskim rzeczywistemu. Gdyby wybrano metrykę Card Precision@100 jako tą decydującą o wyborze detektora, lepszym rozwiązaniem wydaje się być wybranie agregacji z drzewami decyzyjnymi, wspomaganej zmniejszaniem liczebności klasy większościowej w sposób losowy. Tak zbudowany detektor dostarczałby marginalnie lepszych predykcji, będąc jednocześnie znacznie szybszym zarówno w czasie wnioskowania jak i czasie treningu. Natomiast w przypadku wybrania wyniku F1 jako metryki decydującej o użyteczności detektora, detektor LSTM byłby najlepszy − zakładając, że jego czas wnioskowania dla prawie 60 tysięcy transakcji, wynoszący 1.342 sekundy, spełniałby wymagania systemowe. Jeśli nie, należałoby wybrać detektor XGBoost.

6.1.5 Uczenie głębokie a zespołowe z drzewami decyzyjnymi

Jak zostało przedstawione w poprzedniej podsekcji, wybór najlepszej metody detekcji zależeć będzie od tego, jakie metryki ewaluacyjne są najważniejsze oraz czy czas treningu i wnioskowania są kluczowymi czynnikami wpływającymi na decyzję. Zasugerowano wybór czterech detektorów, które należałoby wybrać, w zależności od potrzeb. Dwa z nich, czyli gMLP i LSTM to detektory wykorzystujące uczenie głębokie, natomiast agregacja z drzewami decyzyjnymi i XGBoost to detektory uczenia zespołowego, które wykorzystują drzewa decyzyjne. Z racji, że większość przetestowanych w tej pracy detektorów wykorzystuje uczenie głębokie, powinno się raczej przeanalizować rozmieszczenie detektorów wykorzystujących uczenie zespołowe z drzewami decyzyjnymi wśród najlepszych wyników dla metryk ewaluacyjnych. Dla każdej z metryk, wśród najlepszych pięciu wyników znajduje się detektor wykorzystujący wspomniane uczenie zespołowe, osiągając dla CP@100 i AUC ROC najlepszy wynik. Ponadto, dla każdej z metryk, wśród dziesięciu najlepszych wyników znajdą się trzy lub cztery detektory z uczenia zespołowego. Powyższe obserwacje pokazują zatem, że uczenie zespołowe z drzewami decyzyjnymi radzi sobie równie dobrze co uczenie głębokie, dla wybranego zbioru danych. W [24] autorzy przedstawiają, że metody zespołowe wykorzystujące drzewa decyzyjne osiągają lepsze wyniki od metod uczenia głębokiego, dla danych tabelarycznych. W [18] przeprowadzono zaawansowaną analizę osiąganych wyników przez modele uczenia głębokiego dla problemów klasyfikacji w szeregach czasowych, w której pokazano, że niektóre z modeli potrafią wypaść lepiej w porównaniu do klasycznych metod, jak na przykład las losowy. W [31] zaproponowano zespół różnych sieci neuronowych, aby zbudować solidny model wykrywania anomalii. Porównując wyniki wspomnianego mo-

delu zespołowego różnych sieci neuronowych z detektorem XGBoost na dwóch zbiorach danych pokazano, że XGBoost przewyższa model zespołowy pod względem dokładności (ang. accuracy), precyzji i czułości. Dlatego też, aby móc porównać wyniki uzyskane w wymienionych pracach z wynikami z tej pracy, należałoby przeprowadzić ewaluację na większej liczbie zbiorów danych, dla większej liczby detektorów.

6.1.6 Kontynuacja

W ramach kontynuacji porównania metod detekcji anomalii w szeregach czasowych, dopełniając wspomniane wyżej plany, można by szczególną uwagę poświęcić na użycie większej liczby detektorów i technik wykorzystujących uczenie nienadzorowane i częściowo nadzorowane. Pozwoliłoby to na zwiększenie różnorodności, z szansą na przełamanie dominacji technik uczenia nadzorowanego w wynikach dla metryk ewaluacyjnych wśród detektorów.

6.2 Metody wyjaśniania

6.2.1 Wyzwania związane z testowaniem biblioteki SHAP

Zjawiskiem, które mogło niekorzystnie wpłynąć na użyteczność wartości uzyskanych z SHAP, jest obecność ostrzeżeń przy wywołaniu metody shap_values dla każdego z testowanych detektorów, niezależnie od tego, czy pochodzi on z podręcznika, czy jest zbudowany przy pomocy TSAI. Ostrzeżenia te informują o braku wsparcia dla konkretnego modułu z pod-modułu torch.nn. Podsumowując, w przypadku testowanych detektorów, DeepExplainer nie obsługuje 15 różnych modułów, w tym między innymi Concat, GRU, Add, Transpose, GELU i AdaptiveMaxPool1d. W takich przypadkach DeepExplainer zakłada, że dany moduł jest liniowy i nie zmienia gradientu.

Innym czynnikiem zwiększającym wyzwanie analizy generowanych wartości SHAP jest to, że zakres wartości SHAP, jak również wartość oczekiwana zwracana przez DeepExplainer, różnią się między detektorami, do których odwołuje się podręcznik, a detektorami zbudowanymi przy użyciu TSAI. Powodem jest to, że modele TSAI są zaimplementowane w taki sposób, że w ostatniej warstwie nie używana jest funkcja aktywacji, w przeciwieństwie do detektorów z podręcznika, które to używają sigmoidy. Uśredniając, wartość oczekiwana generowana przez DeepExplainer dla detektorów TSAI wynosi -6.66. Po zastosowaniu sigmoidy liczba -6.66 zamienia się w liczbę 0.001, która jest znacznie bliższa średniej wartości oczekiwanej dla detektorów z podręcznika, która wynosi 0.008. Niestety taka wsteczna konwersja dla wartości SHAP już nie działa, o czym wspomina autor biblioteki w jednym z wątków w repozytorium Github projektu [48]. Dodatkowo, dla detektorów z podręcznika wartości SHAP mieszczą się w podobnym przedziale, podczas gdy detektory z TSAI są znacznie mniej spójne w zakresie wygenerowanych wartości SHAP dla różnych cech wejściowych.

W przypadku chęci interpretacji konkretnych wartości otrzymanych przy użyciu SHAP, w celu zrozumienia, jak bardzo konkretne cechy wpłynęły na predykcję, trzeba wziąć pod uwagę to, że dane używane podczas detekcji są przeskalowane. Skalowanie danych do wspólnej skali lub przedziału jest ważnym krokiem wstępnego przetwarzania dla wielu modeli uczenia maszynowego, a przede wszystkim głębokiego. Precyzyjniej, użyto klasy StandardScaler z biblioteki scikit-learn w celu standaryzacji danych, czyli procesu normalizacji, w którym to wartości dla cech będą miały średnią wartość

oczekiwaną równą zero, a odchylenie standardowe równe jeden. Dlatego też, aby otrzymać wartości SHAP z powrotem w oryginalnej przestrzeni wejściowej, operacja skalowania powinna zostać odwrócona [32].

6.2.2 Porównanie SHAP i TS COIN

Mimo, że SHAP jest szybszy w przypadku mniejszych próbek zestawu treningowego użytych jako dane tła i jednoczesnego wykorzystania GPU do obliczeń, w przypadku większych rozmiarów danych w tle i dla bardziej precyzyjnych obliczeń wartości SHAP, zaprezentowana implementacja TS COIN jest szybsza niż SHAP. Jeśli chodzi zaś o wyniki obu tych metod dla metryki NDCG i stabilności, TS COIN wypada lepiej. Ranking najbardziej wpływowych cech utworzony przy użyciu wartości anormalnego wyniku atrybutu, w porównaniu z rankingiem utworzonym przy użyciu wartości SHAP, jest bardziej zbliżony do faktycznego rankingu cech. Pod względem stabilności, TS COIN uzyskał wyższy średni wynik, jednocześnie przy mniejszym niż SHAP średnim odchyleniu standardowym dla anormalnego wyniku atrybutu.

6.2.3 Kontynuacja

W ramach kontynuacji porównania metod wyjaśniania anomalii w szeregach czasowych, należałoby przeprowadzić testy na większej liczbie zbiorów danych, dla większej liczby metod. TS COIN okazał się być lepszy pod pewnymi względami od algorytmów dostarczanych przez bibliotekę SHAP, lecz nieprzetestowane pozostają takie metody lokalne jak wspomniana w rozdziale 3 LIME, czy też metody skupiające się na konkretnej grupie detektorów jak Grad-CAM [60], pozwalająca wyjaśniać detektory bazujące na sieci CNN. W ramach tej pracy nie poświęcono także większej uwagi przetestowanemu detektorowi XCM (an eXplainable Convolutional neural network for MTS classification), który według autorów architektury, ma dostarczać wiarygodnych i bardziej informatywnych wyjaśnień. Jeśli chodzi zaś o metodę TS COIN, w przedstawionej implementacji dla szeregów czasowych, wykorzystano tę samą metodę nadpróbkowania elementu odstającego, co w oryginalnej pracy. Istnieją natomiast bardziej zaawansowane metody generowania syntetycznych danych o charakterze szeregów czasowych, takie jak TimeVAE [14], TimeGAN [74] czy TTS-GAN [40], które w tym celu wykorzystują generatywne modele uczenia głębokiego.

Rozdział 7
Podsumowanie

W tej pracy przeanalizowany został problem detekcji i wyjaśniania anomalii w szeregach czasowych, co jest trudnym, ale jednocześnie ważnym zadaniem dla wielu aplikacji. Na początku omówiono od podstaw zagadnienie anomalii oraz samego ich wyjaśniania i detekcji. Przedstawiono przykłady aplikacji metod, a także ich użyteczność i wyzwania, które można napotkać podczas ich używania. Wyjaśniono główne sposoby klasyfikacji metod, zarówno detekcji jak i wyjaśniania, i ich główne przypadki użycia. Następnie szczegółowo omówiono metody detekcji anomalii w szeregach czasowych, zaczynając od przedstawienia tych popularnych, często używanych. Opisano również jakie kroki zostały wykonane w celu stworzenia potoku do wykrywania anomalii, który został wykorzystany w ramach tej pracy, wraz z wyjaśnieniem dlaczego wybrane techniki podczas każdego kroku są tymi optymalnymi, w przypadku szeregów czasowych. Omówiono użyty zbiór danych, użytą metodę walidacji, metryki ewaluacyjne, wykorzystane techniki radzenie sobie z niezbalansowanymi zbiorami danych oraz proces wyszukiwania hiperparametrów. Wyjaśniono także zagadnienia związane ściśle z uczeniem głębokim, które pojawiły się w dalszej części pracy. Następnie omówiono popularne metody wyjaśniania anomalii w szeregach czasowych, które są niezależne od modelu, z podziałem na lokalne i globalne. Przedstawiona została również autorska implementacja metody TS COIN, która rozszerza oryginalną metodę COIN o wsparcie dla szeregów czasowych i detektorów sekwencyjnych. Omówiono zmiany i ulepszenia w porównaniu do oryginalnego algorytmu. W rozbudowanym rozdziale o ewaluacji, przedstawione zostały wyniki testów przeprowadzonych dla metod detekcji, uwzględniając wyniki dla wybranych metryk ewaluacyjnych, czas treningu i wnioskowania oraz optymalne hiperparametry znalezione dla detektorów. W drugiej części rozdziału udokumentowano wyniki testów mających za zadanie sprawdzić użyteczność metod TS COIN i SHAP oraz je między sobą porównać. W ramach wspomnianej ewaluacji zbadano otrzymane wyniki odstawania z metody TS COIN oraz wykorzystano otrzymane wartości anormalnego wyniku atrybutu, również z metody TS COIN, w celu porównania jej z wartości otrzymanymi z SHAP, w testach przy użyciu metryk NDCG i stabilności, a na koniec porównano i omówiono czasy wykonywania dla obydwu metod. Następnie, w rozdziale 6, omówiono wyniki dla metod detekcji z poprzedniego rozdziału, starając się jednocześnie odpowiedzieć na postawione pytanie odnośnie najlepszej metody detekcji oraz porównano wyniki detektorów uczenia głębokiego z wynikami detektorów opartych o uczenie zespołowe z wykorzystaniem drzew decyzyjnych. Na koniec wyjaśniono, jakie wyzwania towarzyszyły testowaniu biblioteki SHAP oraz ostatecznie porównano ze sobą metody SHAP i TS COIN w kontekście wyjaśniania anomalii w szeregach czasowych.

Uśredniając wyniki dla wszystkich przetestowanych metryk ewaluacyjnych, detektor gMLP okazał się być najlepszym i najbardziej uniwersalnym detektorem. W wypad-

ku, gdyby zależało nam tylko na jednej z metryk, gMLP należałoby wybrać tylko dla średniej precyzji. Dla każdej z pozostałych metryk ewaluacyjnych, jeśli chodzi o najlepszy wynik, przoduje inny detektor. Dlatego też, przy wybraniu wyniku F1 lub Card Precision @100 jako głównej metryki, należałoby wybrać kolejno detektor LSTM z biblioteki TSAI lub agregację z drzewami decyzyjnymi. Detektory gMLP i LSTM wykorzystują uczenie głębokie, stąd są zdecydowanie wolniejsze jeśli chodzi o czas treningu i wnioskowania. W związku z tym, gdyby średnia precyzja lub wynik F1 miał być główną metryką ewaluacyjną, a osiągane czasy dla detektorów gMLP i LSTM okazałaby się niewystarczające, należałoby wybrać detektor XGBoost, który dla tych metryk osiągnął najlepsze wyniki spośród detektorów niebędących siecią neuronową. Pokazano również, że detektory oparte o uczenie zespołowe z wykorzystaniem drzew decyzyjnych, dla omawianego zbioru danych, osiągnęły zbliżone wyniki do detektorów wykorzystujących uczenie głębokie. Algorytm wyjaśniający DeepExplainer z biblioteki SHAP, w porównaniu do metody TS COIN, uzyskuje lepsze czasy wykonywania w sytuacji, gdy nie zależy nam na większej dokładności obliczonych wartości SHAP. Na ogół jednak chcemy uzyskać bardziej precyzyjne wartości, a wtedy SHAP okazuje się być zdecydowanie wolniejszy od TS COIN. Metoda TS COIN wypada również lepiej, jeśli chodzi o wyniki dla metryki NDCG, a także jest bardziej stablina od SHAP.

Bibliografia

[1] Daniel W. Apley and Jingyu Zhu. Visualizing the effects of predictor variables in black box supervised learning models, 2019.

[2] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate, 2016.

[3] Gustavo Batista, Ronaldo Prati, and Maria-Carolina Monard. A study of the behavior of several methods for balancing machine learning training data. *SIGKDD Explorations*, 6:20--29, 06 2004.

[4] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(Feb):281--305, 2012.

[5] Thomas Bierhance. Do batch sizes actually need to be powers of 2? `https://wandb.ai/datenzauberai/Batch-Size-Testing/reports/Do-Batch-Sizes-Actually-Need-To-Be-Powers-of-2---VmlldzoyMDkwNDQx`, 2022.

[6] Lukas Biewald. Experiment tracking with weights and biases, 2020. Software available from wandb.com.

[7] Szymon Bobek. Intelligible explainable artificial intelligence. Github, 2021.

[8] Niklas Bussmann, Paolo Giudici, Dimitri Marinelli, and Jochen Papenbrock. Explainable machine learning in credit risk management. *Computational Economics*, 57(1):203--216, 2020.

[9] Marcin Chamera. Methods of detecting and explaining anomalies in time series. `https://github.com/MarcinChamera/mgr-anomaly-ts-xai`, 2023.

[10] Tianqi Chen and Carlos Guestrin. XGBoost. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, aug 2016.

[11] Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation, 2014.

[12] Heejeong Choi, Donghwa Kim, Jounghee Kim, Jina Kim, and Pilsung Kang. Explainable anomaly detection framework for predictive maintenance in manufacturing systems. *Applied Soft Computing*, 125:109147, 2022.

[13] Dato. How to evaluate machine learning models: Part 4 – hyperparameter tuning. https://web.archive.org/web/20160701182750/http://blog.dato.com/how-to-evaluate-machine-learning-models-part-4-hyperparameter-tuning, 2016.

[14] Abhyuday Desai, Cynthia Freeman, Zuhui Wang, and Ian Beaver. Timevae: A variational auto-encoder for multivariate time series generation, 2021.

[15] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale, 2021.

[16] Nelly Elsayed, Anthony S, and Magdy Bayoumi. Deep gated recurrent and convolutional network hybrid model for univariate time series classification. *International Journal of Advanced Computer Science and Applications*, 10(5), 2019.

[17] Kevin Fauvel, Tao Lin, Véronique Masson, Élisa Fromont, and Alexandre Termier. XCM: An explainable convolutional neural network for multivariate time series classification. *Mathematics*, 9(23):3137, dec 2021.

[18] Hassan Ismail Fawaz, Germain Forestier, Jonathan Weber, Lhassane Idoumghar, and Pierre-Alain Muller. Deep learning for time series classification: a review. *Data Mining and Knowledge Discovery*, 33(4):917–963, 2019.

[19] Hassan Ismail Fawaz, Benjamin Lucas, Germain Forestier, Charlotte Pelletier, Daniel F. Schmidt, Jonathan Weber, Geoffrey I. Webb, Lhassane Idoumghar, Pierre-Alain Muller, and François Petitjean. InceptionTime: Finding AlexNet for time series classification. *Data Mining and Knowledge Discovery*, 34(6):1936--1962, sep 2020.

[20] Python Software Foundation. *The Python Language Reference*. Python Software Foundation, 2021.

[21] Jerome H. Friedman and Bogdan E. Popescu. Predictive learning via rule ensembles. *The Annals of Applied Statistics*, 2(3), sep 2008.

[22] Philipp Fukas, Jonas Rebstadt, Lukas Menzel, and Oliver Thomas. Towards explainable artificial intelligence in financial fraud detection: Using shapley additive explanations to explore feature importance. In Xavier Franch, Geert Poels, Frederik Gailly, and Monique Snoeck, editors, *Advanced Information Systems Engineering*, pages 109--126, Cham, 2022. Springer International Publishing.

[23] F.A. Gers, J. Schmidhuber, and F. Cummins. Learning to forget: continual prediction with lstm. In *1999 Ninth International Conference on Artificial Neural Networks ICANN 99. (Conf. Publ. No. 470)*, volume 2, pages 850--855 vol.2, 1999.

[24] Léo Grinsztajn, Edouard Oyallon, and Gaël Varoquaux. Why do tree-based models still outperform deep learning on tabular data?, 2022.

[25] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg,

Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357--362, September 2020.

[26] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The elements of statistical learning: data mining, inference and prediction*. Springer, 2 edition, 2009.

[27] Katja Hauser, Alexander Kurz, Sarah Haggenmüller, Roman C. Maron, Christof von Kalle, Jochen S. Utikal, Friedegund Meier, Sarah Hobelsberger, Frank F. Gellrich, Mildred Sergon, Axel Hauschild, Lars E. French, Lucie Heinzerling, Justin G. Schlager, Kamran Ghoreschi, Max Schlaak, Franz J. Hilke, Gabriela Poch, Heinz Kutzner, Carola Berking, Markus V. Heppt, Michael Erdmann, Sebastian Haferkamp, Dirk Schadendorf, Wiebke Sondermann, Matthias Goebeler, Bastian Schilling, Jakob N. Kather, Stefan Fröhling, Daniel B. Lipka, Achim Hekler, Eva Krieghoff-Henning, and Titus J. Brinker. Explainable artificial intelligence in skin cancer recognition: A systematic review. *European Journal of Cancer*, 167:54--69, 2022.

[28] Haibo He and Edwardo A. Garcia. Learning from imbalanced data. *IEEE Transactions on Knowledge and Data Engineering*, 21(9):1263--1284, 2009.

[29] Haibo He and Edwardo A Garcia. Learning from imbalanced data sets. *Knowledge and Data Engineering, IEEE Transactions on*, 21(9):1263–1284, 2009.

[30] Jeremy Howard. fastai, 2020.

[31] Sumaiya Thaseen Ikram, Aswani Kumar Cherukuri, Babu Poorva, Pamidi Sai Ushasree, Yishuo Zhang, Xiao Liu, and Gang Li. Anomaly detection using xgboost ensemble of deep neural network models. *Cybernetics and Information Technologies*, 21(3):175–188, 2021.

[32] insuquot. Dependence plots with unscaled feature values. `https://github.com/slundberg/shap/issues/1396`, 2020.

[33] Andrew Jaegle, Sebastian Borgeaud, Jean-Baptiste Alayrac, Carl Doersch, Catalin Ionescu, David Ding, Skanda Koppula, Daniel Zoran, Andrew Brock, Evan Shelhamer, Olivier Hénaff, Matthew M. Botvinick, Andrew Zisserman, Oriol Vinyals, and João Carreira. Perceiver io: A general architecture for structured inputs & outputs, 2022.

[34] Kalervo Järvelin and Jaana Kekäläinen. Cumulated gain-based evaluation of ir techniques. *ACM Transactions on Information Systems (TOIS)*, 20(4):422--446, 2002.

[35] Joblib Development Team. Joblib: running python functions as pipeline jobs. 2020.

[36] Fazle Karim, Somshubra Majumdar, Houshang Darabi, and Shun Chen. LSTM fully convolutional networks for time series classification. *IEEE Access*, 6:1662--1669, 2018.

[37] Fazle Karim, Somshubra Majumdar, Houshang Darabi, and Samuel Harford. Multivariate lstm-fcns for time series classification. *Neural Networks*, 116:237--245, 2019.

[38] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

[39] Yann-Aël Le Borgne, Wissam Siblini, Bertrand Lebichot, and Gianluca Bontempi. *Reproducible Machine Learning for Credit Card Fraud Detection - Practical Handbook*. Université Libre de Bruxelles, 2022.

[40] Xiaomin Li, Vangelis Metsis, Huangyingrui Wang, and Anne Hee Hiong Ngu. Tts-gan: A transformer-based time-series generative adversarial network, 2022.

[41] Charles X Ling and Victor S Sheng. Cost-sensitive learning and the class imbalance problem. In *Encyclopedia of Machine Learning*, page 153–157. Springer, 2008.

[42] Hanxiao Liu, Zihang Dai, David R. So, and Quoc V. Le. Pay attention to mlps, 2021.

[43] Ninghao Liu. Contextual outlier interpretation. `https://github.com/ninghaohello/Contextual-Outlier-Interpreter`, 2019.

[44] Ninghao Liu, Donghwa Shin, and Xia Hu. Contextual outlier interpretation, 2018.

[45] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation, 2015.

[46] Scott M. Lundberg, Gabriel G. Erion, and Su-In Lee. Consistent individualized feature attribution for tree ensembles, 2019.

[47] Scott M Lundberg and Su-In Lee. A unified approach to interpreting model predictions. In *Advances in neural information processing systems*, pages 4765--4774, 2017.

[48] madsthaks. How to convert logodds explanations to probabilities? `https://github.com/slundberg/shap/issues/963`, 2019.

[49] Branka Hadji Misheva, Joerg Osterrieder, Ali Hirsa, Onkar Kulkarni, and Stephen Fung Lin. Explainable ai in credit risk management, 2021.

[50] Christoph Molnar. *Interpretable Machine Learning*. ?, 2 edition, 2022.

[51] John Muschelli. ROC and AUC with a binary predictor: a potentially misleading metric. *Journal of Classification*, 37(3):696--708, dec 2019.

[52] Ignacio Oguiza. tsai - a state-of-the-art deep learning library for time series and sequential data. Github, 2022.

[53] Sepideh Pashami, Slawomir Nowaczyk, Yuantao Fan, Jakub Jakubowski, Nuno Paiva, Narjes Davari, Szymon Bobek, Samaneh Jamshidi, Hamid Sarmadi, Abdallah Alabdallah, Rita P. Ribeiro, Bruno Veloso, Moamar Sayed-Mouchaweh, Lala Rajaoarisoa, Grzegorz J. Nalepa, and João Gama. Explainable predictive maintenance, 2023.

[54] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, and Lu Fang. Pytorch: An imperative style, high-performance deep learning library. `https://pytorch.org`, 2019. Accessed: [Insert date here].

[55] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825--2830, 2011.

[56] Ismini Psychoula, Andreas Gutmann, Pradip Mainali, S. H. Lee, Paul Dunphy, and Fabien A. P. Petitcolas. Explainable machine learning for fraud detection, 2021.

[57] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. "why should i trust you?": Explaining the predictions of any classifier, 2016.

[58] Takaya Saito and Marc Rehmsmeier. The precision-recall plot is more informative than the roc plot when evaluating binary classifiers on imbalanced datasets. *PloS one*, 10(3):e0118432, 2015.

[59] A. Carlisle Scott, William J. Clancey, Randall Davis, and Edward H. Shortliffe. Explanation capabilities of production-based consultation systems. *American Journal of Computational Linguistics*, pages 1--50, February 1977. Microfiche 62.

[60] Ramprasaath R. Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. Grad-CAM: Visual explanations from deep networks via gradient-based localization. *International Journal of Computer Vision*, 128(2):336--359, oct 2019.

[61] Avanti Shrikumar, Peyton Greenside, and Anshul Kundaje. Learning important features through propagating activation differences, 2019.

[62] Daniel Smilkov, Nikhil Thorat, Been Kim, Fernanda Viégas, and Martin Wattenberg. Smoothgrad: removing noise by adding noise, 2017.

[63] Leslie N Smith. Cyclical learning rates for training neural networks. *arXiv preprint arXiv:1506.01186*, 2017.

[64] Mukund Sundararajan, Ankur Taly, and Qiqi Yan. Axiomatic attribution for deep networks, 2017.

[65] Wensi Tang. Os-cnn. Github, 2020.

[66] Wensi Tang, Guodong Long, Lu Liu, Tianyi Zhou, Michael Blumenstein, and Jing Jiang. Omni-scale cnns: a simple and effective kernel size configuration for time series classification, 2022.

[67] Romain Tavenard, Johann Faouzi, Gilles Vandewiele, Felix Divo, Guillaume Androz, Chester Holtz, Marie Payne, Roman Yurchak, Marc Rußwurm, Kushal Kolar, and Eli Woods. Tslearn, a machine learning toolkit for time series data. *Journal of Machine Learning Research*, 21(118):1--6, 2020.

[68] Robert Tibshirani and Guenther Walther. Cluster validation by prediction strength. *Journal of Computational and Graphical Statistics*, 14(3):511--528, 2005.

[69] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2023.

[70] Jingyuan Wang, Ze Wang, Jianfeng Li, and Junjie Wu. Multilevel wavelet decomposition network for interpretable time series analysis, 2018.

[71] Zhiguang Wang, Weizhong Yan, and Tim Oates. Time series classification from scratch with deep neural networks: A strong baseline, 2016.

[72] Feiyu Xu, Hans Uszkoreit, Yangzhou Du, Wei Fan, Dongyan Zhao, and Jun Zhu. Explainable ai: A brief survey on history, research areas, approaches and challenges. In *Natural Language Processing and Chinese Computing*, pages 563--574, 2019.

[73] Fei Yan, Yunqing Chen, Yiwen Xia, Zhiliang Wang, and Ruoxiu Xiao. An explainable brain tumor detection framework for mri analysis. *Applied Sciences*, 13(6), 2023.

[74] Jinsung Yoon, Daniel Jarrett, and Mihaela van der Schaar. Time-series generative adversarial networks. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[75] George Zerveas, Srideepika Jayaraman, Dhaval Patel, Anuradha Bhamidipaty, and Carsten Eickhoff. A transformer-based framework for multivariate time series representation learning. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, KDD '21, page 2114–2124, New York, NY, USA, 2021. Association for Computing Machinery.

[76] Xiaowu Zou, Zidong Wang, Qi Li, and Weiguo Sheng. Integration of residual network and convolutional neural network along with various activation functions and global pooling for time series classification. *Neurocomputing*, 367:39--45, 2019.

Spis tabel

Spis rysunków